Design, Implementation and Deployment of the *i*KP Secure Electronic Payment System

Mihir Bellare, Juan A. Garay, Ralf Hauser, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, Els Van Herreweghen, Michael Waidner

Abstract— **This paper discusses the design, implementation and deployment of a secure and practical payment system for electronic commerce on the Internet. The system is based on the *i*KP family of protocols – $i = 1, 2, 3$ – developed at IBM Research. The protocols implement credit card-based transactions between buyers and merchants while the existing financial network is used for payment clearing and authorization. The protocols are extensible and can be readily applied to other account-based payment models, such as debit cards. They are based on careful and minimal use of public-key cryptography and can be implemented in either software or hardware. Individual protocols differ in both complexity and degree of security.**

In addition to being both a pre-cursor and a direct ancestor of the well-known SET standard, *i*KP-based payment systems have been in continuous operation on the Internet since mid-1996. This longevity – as well as the security and relative simplicity of the underlying mechanisms – make the *i*KP experience unique. For this reason, this paper also reports on, and addresses, a number of practical issues arising in the course of implementation and real-world deployment of a secure payment system.

Keywords—**Electronic commerce, payment systems, credit & debit cards, multi-party security, public key cryptography.**

Work was done while all authors were with the IBM Research Division.

Mihir Bellare, Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: `mihir@cs.ucsd.edu`.

Juan A. Garay, Bell Labs – Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974, USA. E-mail: `garay@research.bell-labs.com`.

Ralf Hauser, McKinsey & Co, Alpenstr. 3, CH-8065 Zürich, Switzerland. E-mail: `hauser@acm.org`

Amir Herzberg, IBM Research - Haifa Lab (Tel Aviv Office), 2 Weizmann st., Tel Aviv, Israel. Email: `amir@il.ibm.com`,

Hugo Krawczyk, Department of Electrical Engineering, Technion, Haifa 32000, Israel, and IBM T.J. Watson Research Center, New York, USA. Email: `hugo@ee.technion.ac.il`

Michael Steiner, Fachbereich Informatik, Universität des Saarlandes, D-66123 Saarbrücken, Germany, and IBM Zurich Research Laboratory, Rüschlikon, Switzerland. E-mail: `steiner@acm.org`

Gene Tsudik, USC Information Sciences Institute, Marina del Rey, CA 90292, USA. E-mail: `gts@isi.edu`

Els Van Herreweghen, IBM Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland. E-mail: `evh@zurich.ibm.com`,

Michael Waidner, IBM Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland. E-mail: `wmi@zurich.ibm.com`,

Appeared in the IEEE Journal of Selected Areas in Communications, Vol 18, No. 4, April 2000. Part of this work was presented at the First USENIX Workshop on Electronic Commerce, New York, July, 1995

I. Introduction and Overview

At this day and age it is hardly necessary to justify, or stress the importance of, electronic commerce. Suffice it to say that it has been rapidly gaining momentum since early nineties, and has been equally appealing to on-line merchants, consumers and payment providers.

There is a widespread agreement that to enable electronic commerce one needs the means for *secure electronic payments.* Indeed, the appeal of electronic commerce without electronic payment is limited. Moreover, *insecure* electronic payment methods are more likely to impede, than to promote, electronic commerce. Thus, we begin with the premise that security for electronic payments is of the utmost importance.

In this paper we present and discuss a family of secure electronic payment protocols – *i*KP (*i*-Key-Protocol, $i = 1, 2, 3$). These protocols are compatible with the existing card-based business models and payment system infrastructures. They involve three parties: the buyer (who makes the actual payment), the merchant (who will receive the payment) and the acquirer gateway (who acts as an intermediary between the electronic payment world and the existing payment infrastructure, and authorizes transactions by using the latter). Hereafter, we will refer to the acquirer gateway as simply *the acquirer.*

Within this framework we focus on the credit card payment model since it has been the most popular thus far and likely to remain so in the near future. However, other account-based payment models (such as debit cards) are quite similar to the credit card model from technical and security viewpoints, and are thus easily supported by *i*KP.

All *i*KP protocols are based on public-key cryptography, but they vary in the number of parties (out of the three involved) that possess individual public key-pairs and can thus generate digital signatures. This number is reflected in the name of the individual protocols: 1KP, 2KP, and 3KP. The *i*KP protocols offer increasing levels of security and sophistication as the number of parties who possess own public key-pairs increases.

The simplest protocol, 1KP, requires only the acquirer to possess a public key-pair. Buyers and merchants only need to have authentic copies of the acquirer's public key, reflected in a public key certificate. This involves a minimal public key infrastructure (PKI) to provide certificates for a small number of entities, namely, the acquirers. This type of a PKI can be operated, for example, by a large credit card company. In the 1KP setting, buyers are authenticated on the basis of their credit card numbers and optional secret PINs. Payments are authenticated by communicating the credit card number and optional PIN appropriately encrypted under the acquirer's public key, and cryptographically bound to relevant transaction information (purchase amount, identities, etc.). This prevents fraudulent merchants from collecting credit card numbers and creating fraudulent payments.[1] 1KP does not offer non-repudiation for messages sent by buyers and merchants. This means that disputes about the authenticity of payment orders are not unambiguously resolvable within the digital system.[2]

2KP demands that merchants, in addition to acquirers, hold public key-pairs and public key certificates. The protocol can thereby provide non-repudiation for messages originated by merchants. Additionally, 2KP enables buyers to verify that they are dealing with *bona fide* merchants by checking their certificates, without any on-line contact with a third party. As in 1KP, payment orders are authenticated via the buyer's credit card number and PIN, encrypted before transmission.

3KP further assumes that buyers have their own public key-pairs and public key certificates, thus achieving non-repudiation for all messages of all parties involved. Payment orders are authenticated by the combination of credit card number, optional PIN and a digital signature of the buyer. This makes the forging of payment orders computationally infeasible. Additionally, 3KP enables merchants to authenticate buyers on-line. This requires a full public key infrastructure covering all parties involved.

The main reason for designing these three variants was to enable gradual deployment: 1KP requires only a minimal PKI and would have been suitable for immediate deployment at the time it was proposed in early 1995. 2KP requires a PKI covering all merchants, 3KP one covering all merchants and all card holders. Looking back at what actually transpired with the deployment of *i*KP and its descendant, SET, there was actually no need for a 1KP-like protocol.

All *i*KP protocols can be implemented in either software or hardware. In fact, in 1KP and 2KP, the buyer does not even need a personalized payment device: only credit card data and PIN (if present) must be entered to complete a payment. However, for the sake of increased security, it is obviously desirable to use a tamper-resistant device to protect the PIN and – in case of 3KP – the secret key of the buyer.

We emphasize that the goal of *i*KP is to enable *payments.* It is not concerned with any aspect of the determination of the order; it assumes that the order, including price, have already been decided on between buyer and merchant. It does, however, provide secure and unambiguous linking of order information with the payment to enable effective dispute handling.

*i*KP protocols do not provide secrecy (encryption) of the order information. Such protection is assumed to be provided by other mechanisms, e.g., SSL [3] or IPSec [4]. This decoupling of order encryption from the electronic payment protocol is an important design principle of *i*KP which supports compatibility with different underlying browsing and privacy-protecting mechanisms. It also contributes to the overall simplicity, modularity, and ease of analysis of the protocols. An additional advantage is freeing *i*KP from US export restrictions related to the use of bulk encryption. Nonetheless, if desired, the *i*KP family (especially, 2KP and 3KP) can be easily extended to generate shared keys between buyer and merchant for protection of browsing and order information.

The rest of this paper is organized as follows: Section II provides a brief summary of the history of *i*KP and its relation to the current credit card payment standard, SET. The different roles – buyer, merchant, acquirer – are introduced in Section III, and their different security requirements are analyzed in Section IV. The *i*KP family is described and analyzed in Section V. Then, Section VI describes, in detail, the *i*KP implementation architecture and reports on the deployment. The paper concludes with the Appendix A elaborating on the particular technique used for public key encryption in *i*KP.

II. History and Related Work

*i*KP was developed in early 1995 by a group of researchers at the IBM Research labs in Yorktown Heights and Zürich. Right from the beginning the main goal was to work towards an open industry standard. We distributed the *i*KP protocols in the Internet Draft form, invited comments from the scientific community, and presented our design at the Internet Engineering Task Force meeting in Summer of 1995. Subsequently *i*KP was incorporated into the "Secure Electronic Payment Protocols (SEPP)," a short-lived standardization effort by IBM, MasterCard, Europay and Netscape. SEPP, in turn, was a key starting point for "Secure Electronic Payments (SET)," the joint VISA/MasterCard standard for credit card payments [5]. In fact, SET still retains many of the *i*KP-esque features.

Other important ancestors of SET are the "CyberCash Credit Card Protocol" by CyberCash, and the "Secure Transaction Technology (STT)" by Microsoft and VISA. All these ancestors were proposed independently of each other but use similar forms of cryptographic protocols.

From 1994 to 1996 an almost countless number of payment protocols for all kinds of payment models were proposed (see [6] for a survey). One important difference between *i*KP and most of these proposals is that *i*KP was not just a paper design: The "Zurich *i*KP Prototype (Z*i*P)" is a fully operational prototype of 2KP and 3KP. Although it did not become a commercial product, Z*i*P has been successfully deployed in a number of business trials in Europe and Japan since mid-1996. At the time of this writing, a major Dutch payment provider (Interpay Nederland) is still using Z*i*P in their I-Pay system, supporting 80 online merchants and $17,000$ card holders.

[1] Strictly speaking, one cannot consider a number that is given to any restaurant waiter or receptionist a valuable secret, in any sense. But even if the buyer is not liable, knowing his or her credit card number is sufficient to commit certain frauds, and thus overall system security is improved if this number is protected. Moreover, collecting credit card number over the Internet completely changes the scale of the fraud problem; see, e.g., [1].

[2] From a legal point of view such ambiguities are not necessarily a problem – provided there are fixed rules how to resolve them, and all parties are aware of these rules. Some consequences of systems where those rules were not appropriately designed are illustrated in [2].

Another important difference between *i*KP and other credit card payment protocols is its simplicity, modularity, and – to the extent the term can be used – elegance. *i*KP was designed from a small, well-defined set of security requirements (as seen in Section IV), which resulted in a multi-party secure scheme where no party is forced to trust other parties unnecessarily. We focused on the core payment functionality and deliberately omitted all non-payment functionality that could be easily added on top of *i*KP, such as secrecy of order information or fair delivery of goods. Finally we designed *i*KP as a family of protocols, thus anticipating gradual deployment.

Today only two approaches for secure credit card payments over the Internet are practically relevant: SET and encryption of credit card data via SSL [3] or its eventual successor TLS [7]

SET and its ancestor *i*KP, especially 3KP, are very similar. The main difference is in their overall functionality and complexity: *i*KP was designed as a lightweight protocol that provides the core payment functionality only, and is therefore relatively simple to understand and to analyze. SET was designed to support all options that exist in today's credit card operation and is therefore semantically much richer than *i*KP, but also much more difficult to analyze, implement and deploy.

SSL is the *de facto* standard for secure (i.e., encrypted and integrity-protected) client-server communication on the web; it is integrated in virtually all web browsers and servers. SSL uses public-key cryptography, like SET and *i*KP, but typically only servers (i.e., merchants) have public-key certificates while clients (i.e., buyers) are anonymous. Encrypting credit card data with SSL is certainly better than sending them in the clear, but the gain in payment security is very limited:

- For the acquirer the use of SSL is completely transparent – no messages are signed – and thus the merchant does not gain any security.
- SSL does not hide credit card numbers or any other information from the merchant. Thus, it cannot be used for PIN-based authorization.
- Unlike SET or Z*i*P, SSL does not mandate any specific public-key infrastructure. Thus, there is no guarantee that a buyer can verify the merchant's public-key certificate, and even if the certificate can be verified the semantics of such a certificate is not clear. In SET, a special purpose public-key infrastructure can be managed by the credit card organizations.
- In SSL, merchants and acquirers need additional mechanisms (beyond SSL) to transmit credit card and authorization information. *i*KP and SET, in contrast, provide a complete, self-contained solution.

Most types of payment systems, not just credit and debit cards, exist in the digital world. Typically each model requires its own type of protocols, i.e., one cannot expect that *i*KP can be applied to payment models that are very different to the credit card model. We refer to [6] for a survey of other account-based payment models and protocols.

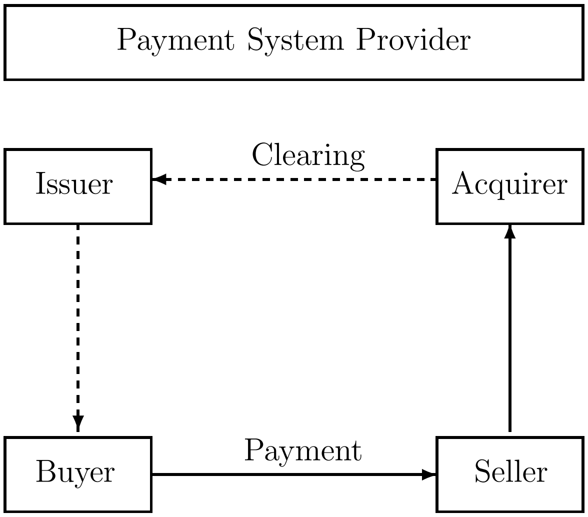


Fig. 1
GENERIC MODEL OF A PAYMENT SYSTEM

III. *i*KP PAYMENT MODEL

PARTIES. All *i*KP protocols are based on the existing credit-card payment system. The parties in the payment system are shown in Figure 1.

The *i*KP protocols deal with the *payment* transaction only (i.e., the solid lines in Figure 1), and therefore involve only three parties, called B – Buyer, S – Seller, and A – Acquirer (gateway). Recall that A is not the acquirer in the financial sense, but a *gateway* to the existing credit card clearing/authorization network. In other words, the function of A is to serve as a front-end to the *current infrastructure that remains unchanged.*

The payment system is operated by a *payment system provider* who maintains a fixed business relationship with a number of banks. Banks act as credit card (account) *issuers* to buyers, and/or as *acquirers* of payment records from merchants (sellers). Each issuer has a Bank Identification Number (BIN) assigned when an issuer signs up with a payment system provider. A BIN is embossed on each credit card included as part of the credit card number. BIN also identifies the payment system provider.

We assume that each *buyer* receives its credit card from an issuer, and is somehow assigned (or selects) an optional PIN as is common in current credit card systems. In 1KP and 2KP, payments are authenticated only by means of the credit card number and the optional PIN (both suitably encrypted), while, in 3KP, a digital signature is used, in addition to the above.

It is also assumed (as is natural in the context of electronic payments) that the buyer is using a computer to execute the payment protocol. Since this computer must receive the buyer's PIN and/or secret signature key, it must be a trustworthy device. We caution that even a buyer-owned computer is vulnerable: it may be used by several people and may contain a Trojan horse or a virus that could steal PINs and secret keys. The best payment device would be a secure isolated and strictly-personal device, e.g., a tamper-resistant smartcard, connected to the computer used for shopping via a buyer-owned smartcard reader with its own keyboard and display. (This is often referred to as an *electronic wallet.*) Technically, 1KP and 2KP can be used with any kind of payment device, while for 3KP the buyers need secure personal devices to store their secret signature keys and certificates.

A *seller* signs up with the payment system provider and with a specific bank, called an *acquirer,* to accept deposits. Like a buyer, a seller needs a secure device that stores the seller's secret keys and performs the payment protocol.

Clearing between acquirers and issuers is done using the existing financial networks.

Public Keys and Certification. Since all *i*KP protocols are based on public key cryptography, a mechanism is needed to authenticate these public keys. We assume a *certification authority, CA*, which has a secret key, SK_{CA}. Its public counterpart, PK_{CA}, is held by all other parties. *CA* will certify a public key of party X by signing the pair (X, PK_X) consisting of the identity of X and X's public key. (The signature is computed under SK_{CA}.) Note that PK_{CA} must be conveyed in an authenticated manner to every party. This is typically done out-of-band, via any of a number of well-known mechanisms.

For simplicity's sake, it is assumed in the rest of the paper that there is a single certification authority. However, it is easy to extend the protocols to support multiple CAs such that the payment system provider at the top-level CA issues certificates to its constituent issuers and acquirers, while these, in turn, issue certificates to their buyers and sellers. The implementation architecture described in Section VI-G supports this model.

In all *i*KP protocols, each acquirer A has a secret key, SK_A, which enables signing and decryption. Its public counterpart, PK_A, which enables signature verification and encryption is held by each accredited seller together with its corresponding *CA*'s certificate. certificate.[3] As in current operation, acquirers receive the buyer's credit card numbers and PINs, and are trusted to keep these values confidential.

Each seller in 2KP/3KP and each buyer in 3KP, has a secret/public key-pair. The public keys are included in certificates issued by the *CA*. The certificates also identify the type of each party (i.e., seller buyer, acquirer.)

Adversaries and Threats. We consider three different adversaries:

- **Eavesdropper:** listens to messages and tries to learn secrets (e.g., credit card numbers, PIN's)
- **Active attacker:** introduces forged messages in an attempt to cause the system to misbehave (e.g., to send him goods instead of to the buyer)
- **Insider:** either a legitimate party or one who learns that party's secrets. (One example is a dishonest seller who tries to get paid without buyer's authorization.)

Before delving into security requirements in Section IV, we briefly discuss common threats and attacks.

The Internet is a decentralized, heterogeneous network, without single ownership of the network resources and functions. In particular, one cannot exclude the possibility that messages between the legitimate parties would pass through a maliciously controlled computer. Furthermore, the routing mechanisms in the Internet are not designed to protect against malicious attacks. Therefore, it is folly to assume either confidentiality or authentication for messages sent over the Internet, unless proper cryptographic mechanisms are employed. To summarize, it is easy to steal information off the Internet. Therefore, at least credit card numbers and PINs must *not* be sent in the clear.

In addition, one must be concerned about the trustworthiness of the sellers providing Internet service. The kind of business that is expected in the Internet includes the so-called *cottage industry* – small sellers. It is very easy for an adversary to set up a shop and put up a fake electronic *store front* in order to get buyers' credit card numbers (e.g., [1]). This implies that the credit card number should travel from buyer to acquirer without being revealed to the seller (who needs only the BIN which can be provided separately.)

Obviously, a good deal of care must be taken to protect the keys of acquirers. One of the biggest concerns is that of an adversary breaking into an acquirer computer through the Internet connection. Therefore, the acquirer's computer must be protected with the utmost care; including a very limited Internet connection using advanced firewall technology (e.g., [8], [9].)

Furthermore, the trust in the acquirer's computer must be limited, so that a break-in would have a limited effect only.

IV. Security Requirements

In this section we consider a range of security requirements for each party involved in the payment process: issuer/acquirer, seller and buyer. They range from mandatory security requirements to optional features.

Issuer/Acquirer Requirements. The issuer and the acquirer are assumed to enjoy some degree of mutual trust. Moreover, an infrastructure enabling secure communication between these parties is already in place. Therefore, we unify their respective requirements.

A1– *Proof of Transaction Authorization by Buyer.* When the acquirer debits a certain credit card account by a certain amount, the acquirer must be in possession of an unforgeable *proof* that the owner of the credit card has authorized this payment. This proof must not be "replayable," or usable as proof for some other transaction. This means it must certify at least the amount, currency, goods description, seller identification, and delivery address, and

[3] The above is somewhat oversimplified since, in practice, an acquirer needs two public key-pairs: one for signatures and the other for encryption. (It is well-known that using the same key-pair for both purposes is not a good idea.) However, for conciseness' sake, this distinction is not reflected in the protocol description below.

be obtained in such a way that replay is not possible. (We use a combination of time stamps and nonces for this purpose). Note also that in this context the seller may be an adversary, and even such a seller must not be able to generate a fake debit. We distinguish between:

(a) *Weak Proof*, which authenticates the buyer to the acquirer but does not serve as a proof for third parties, and

(b) *Undeniable Proof*, which provides full non-repudiation, i.e., can be used to resolve disputes between the buyer and the payment system provider.

The same distinction will be made for all subsequently required proofs of transaction.

A2– *Proof of Transaction Authorization by Seller.* When the acquirer authorizes a payment to a certain seller, the acquirer must be in possession of an unforgeable *proof* that this seller has asked that this payment be made to him.

Seller Requirements.

S1– *Proof of Transaction Authorization by Acquirer.* The seller needs an unforgeable proof that the acquirer has authorized the payment. This includes certification and authentication of the acquirer, so that the seller knows he is dealing with the real acquirer, and certification of the actual authorization information. Note that, again, the amount and currency, the time and date, and information to identify the transaction must be certified. We also distinguish between (a) *Weak proof* and (b) *undeniable proof*, with the latter providing full non-repudiation.

S2– *Proof of Transaction Authorization by Buyer.* Even before the seller receives the transaction authorization from the acquirer, the seller might need an unforgeable proof that the buyer has authenticated it. Again we distinguish between (a) *Weak Proof* and (b) *Undeniable Proof.* This requirement is necessary to provide for *off-line authorization.*

Buyer Requirements.

B1– *Impossibility of Unauthorized Payment.* It must be impossible to charge a buyer's credit card without possession of the credit card number, PIN and, in case of 3KP, the buyer's secret signature key. Thus, neither Internet rogues nor malicious sellers must be able to generate spurious transactions which end up approved by the acquirer. This must remain the case even if the buyer has engaged in many prior legitimate transactions. In other words, information sent in one (legitimate) transaction must not enable a later spurious transaction. So in particular the PIN must not be sent in the clear, and not even be subject to guessing attacks! Similar to the two type of proofs of transactions, we distinguish between:

(a) *Impossibility:* unauthorized payments are impossible provided the acquirer is honest and its secret key is not available to the adversary, and

(b) *Disputability:* buyer can prove not having authorized a payment even if the acquirer's secret key is available to the adversary (e.g., because the adversary colludes with an insider).

We can observe that, in fact, these two requirements are typically met by satisfying the corresponding acquirer requirements A1.a and A1.b, respectively.

B2– *Proof of Transaction Authorization by Acquirer.* Buyer might need to have proof that the acquirer authorized the transaction. This "receipt" from the acquirer is not of paramount importance, but is convenient to have. Again, we distinguish between (a) *Weak Proof* and (b) *Undeniable Proof* (full non-repudiation).

B3– *Certification and Authentication of Seller.* Buyer needs proof that the seller is accredited at some known acquirer (which could be considered as some guarantee for the trustworthiness of the seller).

B4– *Receipt from Seller.* Buyer might need proof that the seller who has previously made the offer has received payment and promised to deliver the goods. This takes the form of undeniable receipt. 2KP and 3KP satisfy this requirement, but will not ensure *fairness* [10], [11]: since the seller can always refuse sending this receipt while already having received the authorization from the acquirer. In this case, the buyer must take the next statement of account as replacement for this receipt.

Additional possible buyer requirements. The following requirements (B5 – B6) may also be desirable. Here we shortly discuss their relation to *i*KP; however, they are not explicitly addressed by the *i*KP protocols.

B5– *Privacy.* Buyers might require privacy of order and payment information. For example an investor purchasing information on certain stocks may not want competitors to know which stocks he is interested in. The privacy of order information and the amount of payment should be implemented independently of the payment protocol, e.g., based on SSL [3].

B6– *Anonymity.* Buyers may also want *anonymity* from eavesdroppers and (optionally) from sellers. Furthermore, buyers may even want anonymity with respect to the payment system provider. *i*KP does not focus on anonymity and, in particular, offers no anonymity from the payment system provider. This might be desirable for systems that aim to imitate cash, but is not essential for protocols, like *i*KP, that follow the credit card-based payment model. However, *i*KP does try to minimize exposure of buyers identities with respect to sellers and outsiders.

V. The *i*KP Protocol Family

In this section we present the three *i*KP protocols, $i \in \{1, 2, 3\}$. We first describe the cryptographic primitives as well as the general protocol structure. The subsequent subsections describe the actual protocols and discuss them with respect to the security requirements outlined above.

Primitives and keys. Figure 2 summarizes the notation for the cryptographic keys held by the various parties, and the cryptographic primitives we will be using. While A's key-pair must enable signature and encryption, all other key-pairs need to enable signatures only. (For simplicity's sake, it is assumed that each party has only one key-pair; in Section VI specialized key-pairs for encryption and signatures will be used). Note that signing and encryption (even when performed with the same key-pair) are *independent* operations; in particular: $\mathcal{E}_X(\mathcal{S}_X(\alpha)) \neq \alpha$.

As reflected in table 2 the encryption function $\mathcal{E}_X$ must provide some form of "message integrity." Thus, a decryption operation results in either a plaintext message or in a flag indicating failure (invalid ciphertext). Formally, the primitive we need is an encryption function secure against adaptive choosen ciphertext attacks. The implication is that correct decryption convinces the decryptor that the transmitter "knew" the plaintext that was originally encrypted. In other words, ciphertext tampering is detectable. Two practical and provably secure schemes achieving this property are: Optimal Asymmetric Encryption Padding (OAEP) [16] (which provides for "plaintext awareness") and Cramer-Shoup [17]. We use the former scheme which we describe in Appendix A.

We stress that plaintext-aware encryption does not provide authentication in the manner of a signature (i.e., no non-repudiation). Nonetheless, it can be made to provide an authentication-like capability between parties sharing a key (such as the BAN or PIN). We also note that the encryption function must be *randomized:* $\mathcal{E}_X$ over message m mixes in a strong randomizing value (a nonce or confounder) such that muliple encryptions of the same plaintext are different and unlinkable.

The *i*KP prototype implementation described in Section VI uses RSA with 1024-bit key length for signatures and for RSA-based plaintext-aware encryption. MD5 is used as both a hash function $\mathcal{H}(\cdot)$ and a primitive in the keyed hash function $\mathcal{H}_k(K, \cdot)$. (The actual keyed hash construct is based on HMAC [14]).

Figure 3 is a list of quantities that will occur in the protocols. Their meaning and usage will be further explained as we go along.

Keys:

$\mathrm{PK}_X, \mathrm{SK}_X$	Public and secret key of Party X ($X =$ Certification Authority CA, Buyer B, Seller S, Acquirer A).
CERT_X	Public key certificate of Party X, issued by CA. We assume it includes X, PK_X and CA's signature on X, PK_X.

All protocols assume A has a public key, and any party needing it has PK_{CA}. 1KP assumes no other keys; 2KP additionally assumes S has a public key; 3KP further assumes B also has a public key.

Cryptographic primitives:

$\mathcal{H}(\cdot)$	A strong collision-resistant one-way hash function which returns strong pseudo-random values. (Examples: MD5 [12], SHA-1 [13])
$\mathcal{H}_k(K, \cdot)$	A one-way hash function requiring, in addition to collision-resistance, no information leakage with respect to its other arguments, if the first argument K is chosen at random. I.e., $\mathcal{H}_k(K, \cdot)$ should behave like a family of pseudo-random functions. (Example: HMAC [14], [15].)
$\mathcal{E}_X(\cdot)$	Public-key encryption with PK_X, performed in a way to provide both confidentiality and some kind of computational "message integrity." (Example: OAEP [16].)
$\mathcal{S}_X(\cdot)$	Signature computed with SK_X. Note that the signature does NOT include the actual message M, i.e., the signature function hashes the message before signing.

Fig. 2

Keys and cryptograhic primitives used in *i*KP protocols

Framework of *i*KP protocols. The protocols have a common framework. Figure 4 illustrates the flows at a very high level. Before the protocol begins, each party $X \in \{A, B, S\}$ has some starting information represented by ST-INF_X. The buyer starts with the public key PK_{CA} of the certification authority. The seller has the certificate CERT_A of the acquirer, and the acquirer has his own certificate CERT_A plus the corresponding secret key SK_A. Each party might also have other information, which differs depending on the specific protocol.

It is assumed that before the protocol starts, the buyer and the seller have agreed on the description and price of the items to buy. The functionality required to shop and agree on the item and price are provided by other means

Quantities occuring in all three protocols:

SALT$_B$	Random number generated by B. Used to salt DESC and thus ensure privacy of order information (DESC) on the S to A link; also used to provide freshness of signatures (Sig$_S$ and Sig$_A$).
AUTHPRICE	Amount and currency.
DATE	Seller's date/time stamp, used for "coarse-grained" payment replay protection
NONCE$_S$	Seller's nonce (random number) used for more "fine-grained" payment replay protection
ID$_S$	Seller id. This identifies seller to acquirer.
TID$_S$	Transaction ID. This is an identifier chosen by the seller which uniquely identifies the context.
DESC	Description of purchase/goods, and delivery address. Includes payment information such as credit card name, bank identification number, and currency. Defines agreement between buyer and seller as to what is being paid for in this payment transaction.
BAN	Buyer's Account Number (e.g., credit card no.).
EXPIRATION	Expiration date associated with Buyer's Account Number.
R$_\mathrm{B}$	Random number chosen by buyer to form ID$_B$. It must be random (not just unique) in order to serve as proof to the buyer that the seller agreed to the payment.
ID$_B$	A buyer pseudo-ID which computed as ID$_B = \mathcal{H}_k(R_B, \mathrm{BAN})$.
RESPCODE	Response from the clearing network: YES/NO or authorization code.

Quantities occuring in some of the protocols:

PIN	Buyer PIN which, if present, can optionally be used in 1KP and 2KP to enhance the security.
SALT$_C$	Random number used to salt the account number in the buyer's certificate.
V	Random number generated by seller in 2KP and 3KP for use as a proof that seller has accepted payment (i.e., to bind Confirm and Invoice messages).
VC	Random number generated by seller in 2KP and 3KP for use as proof that seller has not accepted payment i.e., to bind negative Confirm/Cancel and Invoice messages.

Fig. 3

Definitions of atomic fields used in *i*KP protocols

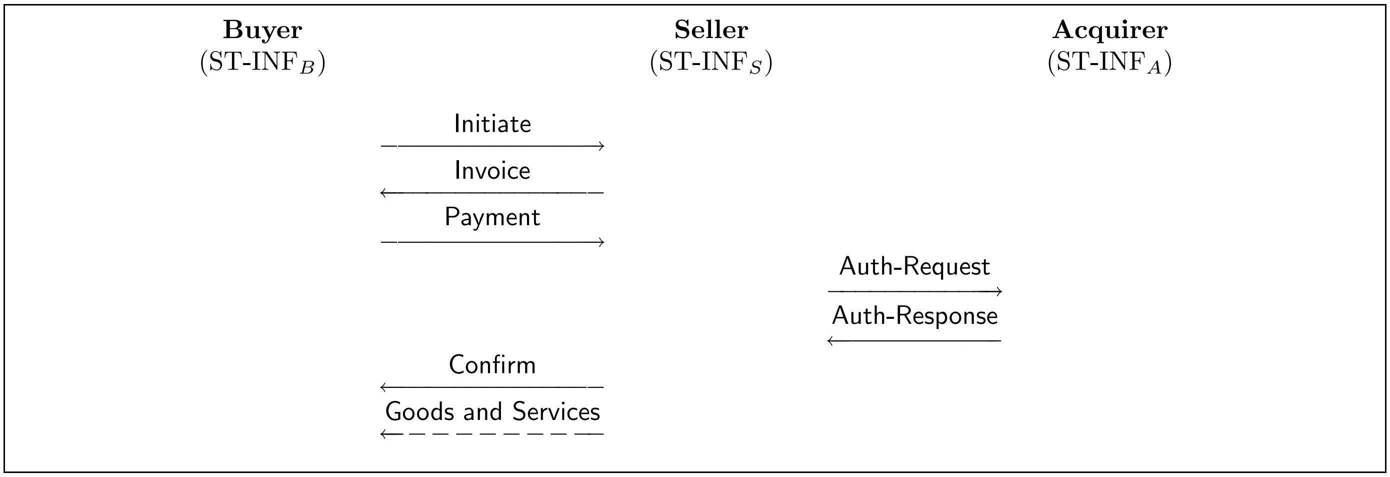


Fig. 4

Framework of *i*KP protocols

(e.g., the web browser).[4] Thus, DESC and AUTHPRICE are part of the starting information of seller and buyer. The basic protocol consists of five flows. (Section VI describes auxiliary flows that provide transaction cancellation, payment clearance, and payment status inquiry.)

The exact content of the flows depends on the actual protocol. At a high level, however, there is a common structure. The buyer starts with the Initiate flow. The seller responds by sending back the Invoice. The buyer then forwards the Payment flow which the seller uses to send an authorization request Auth-Request flow to the acquirer. The acquirer goes through the financial network to obtain authorization and returns the authorization response flow Auth-Response to the seller. Finally, the seller processes this flow and produces a confirmation flow Confirm for the buyer.

The main difference between 1,2 and 3KP is the increasing use of digital signatures as more of the parties involved possess a public/secret key-pair.

A. 1KP

Figure 5 shows the three *i*KP protocols. For the discussion of 1KP, the additional variables and messages added for the 2KP and 3KP protocols ($[_{2,3} \ldots]$ and $[_3 \ldots]$) are ignored. 1KP represents the initial step in the gradual introduction of a public-key infrastructure. Although it requires the use of public-key encryption by all parties, only the acquirer, A, needs to possess and distribute its own public key certificate, CERT_A. In particular, the total number of certificates is small since it is determined only by the number of acquirers.

Like all members of the *i*KP Family, 1KP requires all buyers and sellers to have an authentic copy of PK_{CA}, the public key of the certification authority. Each buyer B has an account number BAN (e.g., a credit card number) and associated EXPIRATION, both known to the payment system. B may also have a secret PIN which is also known (possibly under a one-way function image) to the payment system (but not to sellers). Every seller knows the certificate of its acquirer, CERT_A and, if needed, can send it to the buyer during the course of the protocol. The transport of certificates is not made explicit in our description of *i*KP; where necessary they are assumed to be piggybacked onto *i*KP messages.

1KP does not require A to keep state on a per buyer basis. Buyer's information is verified through the existing authorization infrastructure which uses tamper-resistant technology for processing and verification of PIN's.

All parties in 1KP must perform certain public key computations. Encryption is only performed *once* and only by B, for sending account data (and optional PIN) as part of SLIP. Conversely, decryption is only performed by A (this is also the case for 2KP and 3KP). In 1KP, only A signs data, which is then verified by both B and S. We now turn to the flow-by-flow actions of the parties.

Initiate: Buyer forms ID_B by generating a random number R_B and computing $\mathrm{ID}_B = \mathcal{H}_k(R_B, \mathrm{BAN})$. Buyer generates another random number SALT_B to be used for "salting" the hash of merchandise description (DESC) in subsequent flows. Buyer sends Initiate flow.

Invoice: Seller retrieves SALT_B and ID_B from Initiate, obtains DATE. Generates random quantity (nonce) NONCE_S. The combination of DATE and NONCE_S is used later by A to uniquely identify this payment: the nonce disambiguates payments with common DATE. Seller then chooses transaction id TID_S which identifies the context and computes $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$. Seller forms Common as defined above and computes $\mathcal{H}(\mathrm{Common})$. (Note: seller does not need to additionally "salt" $\mathcal{H}(\mathrm{Common})$ because it contains the already-salted $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$.) Finally seller sends Invoice. CERT_A can be tagged onto this message or sent to B at a later time, e.g., together with the Confirm message.

Payment: Buyer retrieves Clear from Invoice and validates DATE within a pre-defined time skew. B computes $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$. (Note that B already has AUTHPRICE and ID_B and can thus form Common.) He then computes $\mathcal{H}(\mathrm{Common})$ and checks that it matches the value in Clear. Next, B forms SLIP as defined in Figure 5 with PIN being optional. Finally, the slip is encrypted under the acquirer's public key ($\mathrm{EncSlip} = \mathcal{E}_A(\mathrm{SLIP})$) and sent to the seller in the Payment flow.

Auth-Request: The seller now requests the acquirer to authorize the payment. He forwards EncSlip along with Clear and $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$.

Auth-Response: The acquirer extracts Clear, EncSlip and $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ from Auth-Request. A then does the following:

(1) Extracts from Clear: ID_S, TID_S, DATE, NONCE_S and the value h_1 which presumably corresponds to $\mathcal{H}(\mathrm{Common})$. A now checks for *replays*, i.e., makes sure that there is no previously processed request with the same quadruple: ID_S, TID_S, DATE and NONCE_S.

(2) Decrypts EncSlip. If decryption fails, A assumes that EncSlip has been altered (by an adversary or by S) and the transaction is therefore invalid. Otherwise, A obtains SLIP and, from it, extracts AUTHPRICE, h_2 (corresponding to $\mathcal{H}(\mathrm{Common})$), BAN, EXPIRATION, R_B, and, optionally, PIN.

(3) It checks that h_1 and h_2 match; this ensures that buyer and seller agree on the order information (price, identity of seller, etc).

(4) A then re-constructs Common. (It has AUTHPRICE from SLIP. It has ID_S, TID_S, DATE, and NONCE_S from Clear. It can compute $\mathrm{ID}_B = \mathcal{H}_k(R_B, \mathrm{BAN})$ because it has R_B and BAN from SLIP. Finally it has $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ from Auth-Request. Put together, these yield Common.) A then computes $\mathcal{H}(\mathrm{Common})$ and checks that it matches h_1 above.

[4] Indeed, the current consensus is that functions that are beyond payment, such as price negotiation, should be separated, and standardized; e.g., JEPI [18], SEMPER [19], [20].

Composite Fields:

Common	AUTHPRICE, ID_S, TID_S, DATE, NONCE_S, ID_B, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$, $[_{2,3}\, \mathcal{H}(V),\ \mathcal{H}(VC)\,]$
Clear	ID_S, TID_S, DATE, NONCE_S, $\mathcal{H}(\mathrm{Common})$, $[_{2,3}\, \mathcal{H}(V),\ \mathcal{H}(VC)\,]$
SLIP	AUTHPRICE, $\mathcal{H}(\mathrm{Common})$, BAN, R_B, $[\mathrm{PIN}[_3\, \vert \mathrm{SALT}_C\,]]$, EXPIRATION
EncSlip	$\mathcal{E}_A(\mathrm{SLIP})$
Sig_A	$\mathcal{S}_A(\mathrm{RESPCODE}, \mathcal{H}(\mathrm{Common}))$
$[_{2,3}\, \mathrm{Sig}_S\,]$	$\mathcal{S}_S(\mathcal{H}(\mathrm{Common}))$
$[_3\, \mathrm{Sig}_B\,]$	$\mathcal{S}_B(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common}))$

Starting information of parties:

$\mathrm{ST\text{-}INF}_B$	$\mathrm{DESC}, \mathrm{AUTHPRICE}, \mathrm{BAN}, \mathrm{EXPIRATION}, \mathrm{PK}_{CA}, [\mathrm{PIN}],\ [_3\, \mathrm{SK}_B, \mathrm{CERT}_B, [\mathrm{SALT}_C]\,]$
$\mathrm{ST\text{-}INF}_S$	$\mathrm{DESC}, \mathrm{AUTHPRICE}, \mathrm{PK}_{CA}, \mathrm{CERT}_A, [_{2,3}\, \mathrm{SK}_S, \mathrm{CERT}_S\,]$
$\mathrm{ST\text{-}INF}_A$	$\mathrm{PK}_{CA}, \mathrm{SK}_A, \mathrm{CERT}_A$

Protocol Flows:

Initiate: $B \xrightarrow{\mathrm{SALT}_B,\ \mathrm{ID}_B} S$

Invoice: $B \xleftarrow{\mathrm{Clear},\ [_{2,3}\, \mathrm{Sig}_S\,]} S$

Payment: $B \xrightarrow{\mathrm{EncSlip},\ [_3\, \mathrm{Sig}_B\,]} S$

Auth-Request: $S \xrightarrow{\mathrm{Clear},\ \mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC}),\ \mathrm{EncSlip},\ [_{2,3}\, \mathrm{Sig}_S,\ [_3\, \mathrm{Sig}_B\,]\,]} A$

Auth-Response: $S \xleftarrow{\mathrm{RESPCODE},\ \mathrm{Sig}_A} A$

Confirm: $B \xleftarrow{\mathrm{RESPCODE},\ \mathrm{Sig}_A,\ [_{2,3}\, V \vert VC\,]} S$

Fig. 5
*i*KP Protocols

(5) Next, the acquirer uses the credit card organization's existing clearing and authorization system to obtain on-line authorization of this payment. This entails forwarding: BAN, EXPIRATION, PIN (if present), price, etc., as dictated by the authorization system. Upon receipt of a response RESPCODE from the authorization system, A computes a signature, using the function $\mathcal{S}_A$, on RESPCODE and $\mathcal{H}(\mathrm{Common})$.

Finally A sends Auth-Response to S. TID_S can be tagged onto the message enabling the seller to easily recover the transaction context.

Confirm: The seller extracts RESPCODE and the acquirer's signature from Auth-Response. He then verifies the acquirer's signature and forwards both RESPCODE and Sig_A to the buyer.

1KP satisfies the following requirements:

A1(a) *Proof of Transaction Authorization by Buyer.* SLIP includes the BAN, EXPIRATION and the PIN. (The last one, if present, is known only to the buyer and payment system and is the basis of the buyer's security. If PIN is not present, one must assume the BAN and EXPIRATION are not known to the adversary.) Since B knows PK_{CA} and verifies CERT_A, it is ensured that B does not unwittingly send the BAN, EXPIRATION and PIN to a non-authorized party. A decrypts and checks that the BAN, EXPIRATION and PIN are correct. The chosen-ciphertext security (or plaintext-awareness) of the encryption (see beginning of Section V) implies that SLIP originated with the BAN and PIN holder. An adversary not knowing the BAN or PIN can neither create a fake SLIP nor modify the encryption of a legitimate one to its advantage.

Note that PIN-based authentication provides only a weak proof whereas signature-based authentication (as used in

3KP) provides undeniable proof. Moreover, the probability of guessing the correct PIN is much higher than the probability of guessing a valid signature.

However, the security of the encryption against chosen message attacks (and in particular the fact that it is randomized) implies security against *dictionary attacks*. Note that had $\mathcal{E}_A(\cdot)$ been a deterministic encryption function, an attacker who knows all data fields within SLIP (except PIN) could compute encryptions $\mathcal{E}_A(\text{SLIP})$ for *all* possible values of PIN and determine the correct PIN by comparing all encryptions with the one produced by B. In [21] it is formally proven that off-line dictionary attacks against the PIN number are provably prevented if one uses an encryption function which is semantically secure against chosen ciphertext attacks. (Interestingly, they show that semantic security against plaintext-only attacks is not sufficient to ensure such resistance to dictionary attacks.)

It is important to stress that the "transaction" for which we need proof includes the item description, and in particular the delivery address. It should be impossible for an adversary to divert a legitimate payment by changing the delivery address. The value $\mathcal{H}(\text{Common})$ is included in A's authorization in order to prevent such attacks. In particular, it prevents a special kind of *person-in-the-middle* attack that we now describe.

An attacker who impersonates a seller can get the buyer to agree to purchase something for a given amount. The attacker then obtains from the buyer the encrypted slip authorizing payment. The attacker now impersonates the buyer to the seller, but this time the adversary buys for the same amount a (possibly) different merchandise with different delivery address and "pays" for it with the buyer's slip. Notice, however, that in this case there will be a mismatch between the view of the "order" by the real buyer and the seller, and, consequently, a mismatch in the value of $\mathcal{H}(\text{Common})$.

Finally, replay of SLIP by a dishonest seller is detected by the combination of DATE and NONCE_S. There is an "acceptable delay" period T_{delay}. All SLIP-s are cached for T_{delay} time units past DATE. Different SLIP-s with the same DATE are disambiguated by nonces.

S1(b): *Proof of Transaction Authorization by Acquirer.* The unforgeable, undeniable non-repudiable proof is the digitally-signed message sent by A. The inclusion of $\mathcal{H}(\text{Common})$ prevents replay of authorization messages which would otherwise result in fake authorization of buyer's payments.

Since the seller knows Common in advance, the signature would accurately reflect any tampering in the information sent from seller to acquirer and any disagreement between buyer and seller as far as payment data.

The inclusion of $\mathcal{H}(\text{Common})$ in both the buyer-generated SLIP and (explicitly) in Auth-Request enables A to detect any conflicts between the seller's and buyer's views of the order contents (prior to submitting the transaction to the clearing network).

B1(a): *Unauthorized Payment is Impossible.* This is a direct consequence of satisfying **A1(a)**.

B2(b): *Proof of Transaction Authorization by Acquirer.* This is a direct consequence of satisfying **S1**(b).

B5: *Privacy.* 1KP provides partial privacy. Specifically, acquirer is not given DESC, but only $\mathcal{H}_k(\text{SALT}_B, \text{DESC})$. Furthermore, acquirer (or an eavesdropper on the acquirer-to-seller link) cannot obtain DESC via dictionary attack, for the following reason.

In a dictionary attack, the attacker who has a set of possible values of DESC wants to determine whether one of them corresponds to what the buyer is ordering. If "salt" was not used, the attacker could easily make the check by evaluating $\mathcal{H}$ on collected values and seeing whether one of the results matches $\mathcal{H}(\text{DESC})$ in the subject flow. However, without the knowledge of SALT_B used in $\mathcal{H}_k(\text{SALT}_B, \text{DESC})$ all possible description guesses DESC' will have the same likelihood due to the pseudo-random nature of $\mathcal{H}_k()$.

Of course, an attacker who eavesdrops on both buyer-seller and seller-acquirer links can record SALT_B (transmitted in the clear on the former) and mount a dictionary attack. However, recall that order privacy is not one of our central goals and, if deemed to be a strong concern, buyer-seller communication may be protected by complementary means, such as SSL [3].

1KP also hides buyer's identification information by only disclosing the one-time randomized pseudo-identity ID_B. This quantity is sufficient for the acquirer to bind the transactions to the actual buyer but it does not allow for disclosing of actual ID information (such as the BAN) in public (including to sellers) or to allow an attacker to link different purchases by the same buyer.

The last protocol flow carrying signed authorization by the acquirer is optional. Its only purpose is to provide a transaction receipt for the buyer but is has no impact on the security of the protocol.

To summarize, 1KP is a simple and fairly efficient protocol. Its main achievement was (at the time of its design, circa 1995) to get a secure electronic payment system with as little modification as possible to the existing infrastructure. Its main weaknesses are: 1) the buyer authenticates itself via the acquirer by means of only account number and optional PIN (as opposed to strong authentication via digital signatures); 2) the seller does not directly authenticate itself to the buyer or acquirer (there is however some level of indirect authentication via the buyer's SLIP and the authorization by the acquirer); and 3) neither seller nor buyer provide undeniable receipts for the transaction. Enhancing 1KP to provide these missing features results in the protocols described in the next two subsections, 2KP and 3KP.

B. 2KP

The second protocol, 2KP, is obtained by including also the values and fields specific to 2KP ($[_{2,3} \ldots]$) to the protocol in Figure 5. The basic difference with respect to 1KP is that, in addition to A, each seller S needs to possess a public key with a matching secret key, and distribute its own public key, with its certificate, CERT_S.

We now describe the additions to the flows and actions. There are two new elements in Invoice. The first is that the seller chooses random values V and VC and puts $\mathcal{H}(V)$ and $\mathcal{H}(VC)$ in Clear. (The inclusion of V or VC in Confirm will later serve as a (one-time) "signature," thereby saving the seller one signature computation. See below.) These values will be added to Common for what follows. Second, the seller signs $\mathcal{H}(\mathrm{Common})$ and includes this signature Sig_S in Invoice. Upon receipt of Invoice the buyer checks the seller's signature, and then proceeds as before to generate Payment. Auth-Request is augmented by the seller to include the same signature Sig_S he sent to the buyer earlier. The acquirer checks this signature before authorizing payment. Finally, the value V (success) or VC (failure) is included by the seller in Confirm. The buyer computes $\mathcal{H}(V)$ (or $\mathcal{H}(VC)$) and checks that it matches the value sent earlier in Invoice.

2KP satisfies all the requirements addressed by 1KP, as well as:

A2: *Proof of Transaction Authorization by Seller.* This is achieved by the inclusion of the seller's signature Sig_S and the acquirer's verification of it.

B3: *Certification and Authentication of Seller.* Similarly achieved by inclusion of signature of seller and its check by buyer.

B4: *Receipt from Seller.* This is achieved by the combination of S's signed message sent to A, and the value V (VC) sent in Confirm. V (VC) assures the buyer (and any third party) that the seller has accepted (rejected) the payment. This is because no other party is capable of finding V (VC). (It would require inverting the one-way function $\mathcal{H}$ on the point $\mathcal{H}(V)$ or $\mathcal{H}(VC)$). In this way the cost of an extra digital signatures (such as RSA) is saved.
Obviously, S can refuse forwarding A's authorization message to the buyer and sending its last message. In this case, B does not know whether the transaction was aborted or finalized. This must be handled based on the next account statement.

C. 3KP

The last protocol—3KP—is obtained by including in Figure 5 the fields that are 2KP- and 3KP-specific ($[_{2,3} \ldots]$ and $[_3 \ldots]$, respectively.). As can be expected, in 3KP all protocol participants, including buyers, possess a public key with the associated secret key and certificate. All parties are now able to provide non-repudiation.

CERT_B, sent (out-of-band) to the seller in addition to the *i*KP flows, may contain some additional data besides the buyer's public key and ID. These data are included in the certificate *in salted hashed form* using $\mathcal{H}_k()$. This allows to open the information only on demand and prevents the leaking of information to unauthorized users. For instance, CERT_B might include the hash of the buyer's physical address, and whether ordered goods should be sent to B's home address, B can reveal "Buyer's physical address" and the corresponding salt to the seller who can verify it based on CERT_B. Similarly, CERT_B can securely link the BAN to the signing key. This allows the acquirer to efficiently verify that the payer has the necessary authority over BAN contained in the SLIP (see [15]). 3KP certificates as implemented in the Z*i*P prototype (Section VI) contain a salted hash $\mathcal{H}_k(\mathrm{SALT}_C, \mathrm{BAN})$ of the BAN. Since SALT_C is included in EncSlip, the acquirer can do this check.

The buyer's signature serves as undeniable proof of transaction (A1.b), and enables disputability (B1.b). If the 3KP certificate contains no cleartext buyer identity (e.g., only the above hashed data or possibly a pseudonym), 3KP does not provide the seller with more information about the buyer than 1KP or 2KP. On the other hand, the sellers can link all payments of the buyer with CERT_B and B's signature, i.e., the buyer loses some privacy when compared to 1KP and 2KP. One way to avoid this is by encrypting CERT_B and the signature with A's public key. In that case, the seller of course cannot verify Sig_B but can still rely on Sig_A for a guarantee of the transaction outcome.

Notice that in 3KP PIN numbers can still be used, but only for compatibility with the existing infrastructure. Except for that reason, PINs can be omitted since the level of authentication provided by the buyer's signature is significantly superior to that provided by a PIN. (Note, however, that the inclusion of a user-memorizable PIN can still provide some defense in the case that the buyer's signature key is stolen.)

3KP satisfies all the requirements addressed by 2KP, as well as:

A1(b): *Undeniable Proof of Transaction Authorization by Buyer.* The buyer signs the SLIP using a secret key SK_B known to B only.

S2(b): *Proof of Transaction Authorization by Buyer.* Based on B's signature, S can verify that SLIP was signed by B. S cannot verify the correctness of the contents of SLIP, especially not of the PIN.

B1(b): *Unauthorized Payment is Impossible.* Follows from A1.b.

D. Comparison of the *i*KP protocols

The *i*KP protocols presented above vary in the degree of both protection and complexity. They proceed in an incremental path towards electronic payment with strong security features with respect to all parties involved. Practically speaking, it was envisaged at the time of the design that 1KP would represent a short-term, interim step towards payment protocols with stronger security guarantees. Thereafter, 2KP and 3KP could be gradually phased in. Table V-D presents a comparison of the *i*KP protocols.

The *i*KP family can fulfill all stated requirements and, in particular, provides non-repudiable receipts from the acquirer to the seller/buyer, and from the seller to the buyer. In case that the buyer also possesses a public key-pair (3KP), buyer payment non-repudiation becomes possible.

Although anonymity is not the focus of *i*KP, both 1KP and 2KP provide anonymity of the buyer to the seller, and to the outside: the buyer uses a pseudo-identity ID_B which

TABLE I

Comparison of the *i*KP payment protocols. A requirement marked by $\surd$ is satisfied but not disputable (weak), while $\surd\surd$ indicates that the requirement is satisfied based on undeniable proof (strong) providing non-repudiation and disputability.

REQUIREMENTS/PROTOCOLS	1KP	2KP	3KP
Issuer/Acquirer			
A1. Proof of Transaction Authorization by Buyer	$\surd$	$\surd$	$\surd\surd$
A2. Proof of Transaction Authorization by Seller		$\surd\surd$	$\surd\surd$
Seller			
S1. Proof of Transaction Authorization by Acquirer	$\surd\surd$	$\surd\surd$	$\surd\surd$
S2. Proof of Transaction Authorization by Buyer			$\surd\surd$
Buyer			
B1. Unauthorized Payment is Impossible	$\surd$	$\surd$	$\surd\surd$
B2. Proof of Transaction Authorization by Acquirer	$\surd\surd$	$\surd\surd$	$\surd\surd$
B3. Certification and Authentication of Seller		$\surd\surd$	$\surd\surd$
B4. Receipt from Seller		$\surd\surd$	$\surd\surd$

is different in each transaction and unlinkable across transactions. The buyer's payment activity is thus unlinkable and untraceable. 3KP clearly leaks identity information through the use of buyer certificates. However, through the use of pseudonyms, buyers can remain anonymous, if not unlinkable.

Order privacy against eavesdroppers can be attained by employing a secure communication protocol (e.g., [3], [4]). Alternatively, the *i*KP protocols themselves could be extended to provide this type of protection. Since *i*KP aims at credit card-like payments, no anonymity against the payment system is provided.

As shown in the next section, the *i*KP protocols can be easily extended to support batch processing of payments from the same buyer by the seller and/or to guarantee (block) payment amounts as commonly done, for example, in the case of hotel or car rentals payments. Another avenue for extensions are micro-payments: the relatively high cost of credit card transactions makes *i*KP unsuitable for payments of very small amounts. However, Hauser et al. [22] show how *i*KP can be extended to support micro-payments efficiently without sacrificing strong multi-party security.[5]

The *i*KP protocols were implemented at the IBM Zürich Research Lab in early 1996. The implementation and deployment of the resulting system, referred to as Z*i*P (Zürich *i*KP Prototype), is described in the next section.

VI. Z*i*P: Implementation and Deployment

A. Protocol scenarios

The payment authorization core of the Z*i*P implementation is formed by the 2KP and 3KP protocols described in the previous section. Additional functionality was added during the implementation phase following the requests of the target user community. The actual Z*i*P protocol suite includes four protocol scenarios:

1. Payment Authorization (2KP and 3KP augmented with cancellation option)
2. Separate Payment Clearance (Capture)[6]
3. Refunds
4. Inquiry

B. Payment Authorization

Z*i*P payment authorization consists of the basic payment scenario described in Section V and Figure 5. The Z*i*P implementation (Figure 6) is augmented with an optional Cancel flow (carrying the seller's cancellation commitment VC) which is used by the seller if he decides, for whatever reason, not to go ahead with the authorization request. RESPCODE, in this case, is set by the seller rather than by the acquirer.

The Z*i*P protocols were also augmented with a number of timestamps enabling efficient replay detection and transaction lifetime management: the acquirer sets a timestamp of authorization in AUTHTIME; and the seller specifies an invoice expiration INVOICEEXP. In addition, Z*i*P allows for optional data, OPTSIG_Z, which is not carried in *i*KP flows, to be included in the signatures (see also Section VI-F).

The variable PFLAGS contains a number of protocol flags set jointly by buyer and seller:

- PFLAGS:SIG_B - Buyer's signature Sig_B present in Payment and Auth-Request.

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

A seller can refuse to issue an Invoice if it is the seller's policy to always require Sig_B and the buyer is not able to provide it.

[5] Note that most other micro-payment protocols such as Millicent [23] and NetBill [24] gain their efficiency through the use of shared-key cryptosystems and therefore require complete trust in the payment system provider.

[6] The terms "clearance" and "capture" are used interchangeably throughout this paper.

Composite Fields:

Common	PFLAGS, AUTHPRICE, ID_S, TID_S, DATE, INVOICEEXP, NONCE_S, ID_B, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$, $\mathcal{H}(V)$, $\mathcal{H}(VC)$
Clear	PFLAGS, ID_S, TID_S, DATE, NONCE_S, $\mathcal{H}$(Common), INVOICEEXP, $\mathcal{H}(V)$, $\mathcal{H}(VC)$
SLIP	AUTHPRICE, $\mathcal{H}$(Common), BAN, R_B, $[\mathrm{PIN}\vert\mathrm{SALT}_C]$, EXPIRATION
EncSlip	$\mathcal{E}_A(\mathrm{SLIP})$
Sig_A	$\mathcal{S}_A(\mathrm{RESPCODE}, \mathrm{AUTHTIME}, \mathcal{H}(\mathrm{Common}), \mathrm{OPTSIG}_A)$
Sig_S	$\mathcal{S}_S(\mathcal{H}(\mathrm{Common}), \mathrm{OPTSIG}_S)$
Sig_B	$\mathcal{S}_B(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common}), \mathrm{OPTSIG}_B)$

Protocol Flows:

Initiate:	B	$\xrightarrow{\mathrm{PFLAGS}^a, \mathrm{SALT}_B,\ \mathrm{ID}_B}$	S
Invoice:	B	$\xleftarrow{\mathrm{Clear}, [\mathrm{Sig}_S]}$	S
Payment:	B	$\xrightarrow{\mathrm{EncSlip}, [\mathrm{Sig}_B]}$	S
Cancel:	B	$\xleftarrow{\mathrm{RESPCODE},\ \mathrm{VC},\ [\mathrm{Sig}_S]}$	S
Auth-Request:	S	$\xrightarrow{\mathrm{Clear},\ \mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC}),\ \mathrm{EncSlip},\ \mathrm{Sig}_S,\ [\mathrm{Sig}_B]}$	A
Auth-Response:	S	$\xleftarrow{\mathrm{RESPCODE},\ \mathrm{AUTHTIME},\ \mathrm{Sig}_A}$	A
Confirm:	B	$\xleftarrow{\mathrm{RESPCODE},\ \mathrm{AUTHTIME},\ \mathrm{V}\vert\mathrm{VC},\ [\mathrm{Sig}_A],\ [\mathrm{Sig}_S]}$	S

[a] PFLAGS in Initiate contains all the options set by the buyer; seller may add PFLAGS:CLRN to the final PFLAGS sent in Clear.

Fig. 6
Z*i*P Payment Authorization

- PFLAGS:SIG_S - Sellers's signature Sig_S requested in Invoice.

If Sig_S is not present in Invoice, it has to be present in Confirm (or Cancel) and checked by the buyer at that time. This option is set by the buyer. A buyer may, e.g., gather Sig_S's from multiple sellers for the purpose of comparative shopping, each time suspending the transaction after having received the signed invoice. Such an abbreviated protocol run can be resumed at a later time provided that Sig_S is still timely/valid.

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.

- PFLAGS:CONFIRM - Confirm is requested.

It is set by the buyer; it is envisaged that every seller should support this option. If PFLAGS:CONFIRM is set but the seller delays authorization or cannot reach the acquirer, the seller can reply with a Status flow (see Section VI-E).

- PFLAGS:SIG_A - Sig_A is requested in Confirm.

It is set by the buyer and can only be used in conjunction with the PFLAGS:CONFIRM option.

- PFLAGS:CLRN - Authorization and clearance (capture) are performed together.

This option is set by the seller. If set, the protocol in Figure 6) suffices to complete payment. Otherwize, the protocol only achieves payment authorization; the seller must subsequently perform a seperate clearance function (Section VI-C) or take further processing of this payment off-line. The response code from the acquirer in Auth-Responseindicates whether authorization is given, and, if clearance was requested, whether the payment was cleared.

While a buyer may, in principle, refuse this option, this is not likely.

- PFLAGS:noEnc - The buyer does NOT use encryption.

SLIP is sent in the clear and contains neither a PIN nor SALT_C. This flag is set by the buyer. Sellers have no say over this option. The purpose is to avoid the expense of encryption and to satisfy certain export regulations in cases when BAN's are not treated as secret or sensitive information. Can only be used in conjunction with the PFLAGS:SIG_B option.

C. Separate Payment Clearance/Capture

The protocol shown in Figure 7 performs separate clearance after a prior successful payment authorization with PFLAGS:CLRN not set. This clearance must be performed with the same acquirer that handled the previous authorization.

Multiple clearance flows against the same payment authorization are supported. Replay of clearance requests is checked by the acquirer based on the CLRNSEQ counter. CLRNPRICE represents the total amount cleared; if CLRNPRICE is lower than in the previous Clrn-Request, this is a request for refunds (Section VI-D). Including the total cleared price (as opposed to incremental differences) makes the system more robust against inconsistencies between seller and acquirer due to loss of messages; also, it allows for more efficient state-keeping and proof-collection (both seller and acquirer only have to store the last Clrn-Request and/or Clrn-Response).

D. 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 *i*KP/Z*i*P).

Within *i*KP/Z*i*P, a refund transaction – for all practical purposes – is equivalent to (and treated as) a clearance/capture transaction. As explained above, a refund is, essentially, a clearance with the lower amount. The difference between a refund and a clearance manifests itself only within the domain of the financial clearing network.

E. Inquiry

The buyer can ask the seller about the status of a specific payment. The protocol is shown in Figure 8. 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 the financial institutions. The response from the seller is either Confirm (if the seller has received Auth-Response), a Status message with his view on the current transaction state (if he hasn't received Auth-Response), or Cancel (if he decided to cancel the payment and hasn't previously sent Auth-Request to the acquirer nor Confirm to the buyer).

F. Implementation rationales and explanations

This section explains some of the features of the Z*i*P protocol design.

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

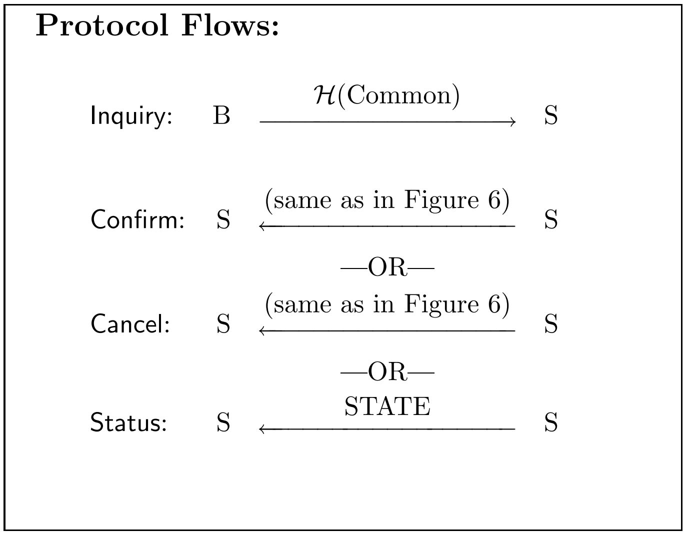

Protocol Flows:
Inquiry: B $\xrightarrow{\mathcal{H}(\text{Common})}$ S
Confirm: S $\xleftarrow{\text{(same as in Figure 6)}}$ S
—OR—
Cancel: S $\xleftarrow{\text{(same as in Figure 6)}}$ S
—OR—
Status: S $\xleftarrow{\text{STATE}}$ S

Fig. 8
Z*i*P: Inquiry protocol

- 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.
- All fields of the form OPTSIG_Z are (as the name suggests) optional. They carry optional data and are included in the respective Sig_Z signatures.

  For example, they can be used to carry credentials/certificates of various entities (hence their absence from the Z*i*P protocol flows), or other optional data like periodic account statements or additional information returned by the clearing network (such as additional authorization codes needed for separate clearing).

Replay Detection. Since both TID_S as well as buyer-chosen random values serve as input to $\mathcal{H}$(Common), association management and replay detection by buyer and seller Transaction Layers (see Section VI-G) is largely based on $\mathcal{H}$(Common). (An exception is Initiate: since this flow doesn't contain $\mathcal{H}$(Common), a buyer transaction identifier TID_B is tagged onto Initiate by the Transaction Layeras the replay detection key for this flow.)

Similarly, the Transaction Layer at the acquirer uses $\mathcal{H}$(Common) as a transaction identifier. Note that the acquirer and/or financial network must perform replay detection only if the seller requests payment capture processing, to ensure that old Clrn-Request messages cannot be used to change the captured amount. Replay detection for authorization-only requests is a policy matter determined by the individual payment system providers.

Typing of message fields. Every signature type generated in Z*i*P 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.

Composite Fields:

SigClrn_S	$\mathcal{S}_S(\mathcal{H}(\text{Common}),\text{CLRNPRICE, OPTSIGCLRN}_S\text{, CLRNSEQ})$
SigClrn_A	$\mathcal{S}_A(\text{RESPCODE, CLRNTIME, CLRNPRICE, CLRNSEQ, }\mathcal{H}(\text{Common})\text{, OPTSIGCLRN}_A)$

Protocol Flows:

Clrn-Request: $\text{S} \xrightarrow{\mathcal{H}(\text{Common}),\text{CLRNPRICE, CLRNSEQ, SigClrn}_S} \text{A}$

Clrn-Response: $\text{S} \xleftarrow{\text{RESPCODE, CLRNTIME, CLRNSEQ, SigClrn}_A} \text{A}$

Fig. 7
Z*i*P: Clearance protocol

The following distinct signature types are identified: Sig_B, Sig_S, Sig_A, SigClrn_S, SigClrn_A.[7]

The following hash function computations are uniquely identified: $\mathcal{H}(\text{Common})$, $\mathcal{H}(V)$, $\mathcal{H}(VC)$, $\mathcal{H}_k(R_B, \text{BAN})$ and $\mathcal{H}_k(\text{SALT}_B, \text{DESC})$.

Performance. Cryptographic operations such as computation/verification of public key signatures and en/decryption are computationally expensive. *i*KP and Z*i*P are designed to improve performance by minimizing the number of cryptographic operations.

Adherence to export regulations. 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 A, only SLIP is encrypted, and the data therein is limited to financial information.

G. Architecture

Figure 9 shows the architecture of Z*i*P. Buyer, Seller and Acquirer applications, residing on different network nodes, access the *i*KP functionality through the Transaction Layer interface [25]. The Transaction Layer provides the payment applications with a high-level (C++) interface using simple payment objects, such as a BuyerTransaction class with methods Initiate(), Pay(), Inquiry(). The Transaction Layer takes care of association management, audit and configuration. The assocation management finds transactions matching incoming messages based on transaction ids (or $\mathcal{H}$(Common), as described in Section VI-F). It detects duplicate messages and unexpected messages, i.e., those not corresponding to an outstanding or recorded transaction. Unexpected requests are acknowledged with an error message, unexpected replays are ignored.

The Transaction Layer realizes robustness of the Z*i*P protocols in the presence of failures. It keeps transaction state in persistant storage and recovers from system crashes. To overcome unreliable communication, the Transaction Layer tries retransmitting after appropriate timeouts. The replay detection in the Transaction Layer will catch this and correspondingly resend the previous reply. (Note that the messages are idempotent!)

The Comm module sends *i*KP messages between different Z*i*P users. It supports several underlying transport mechanisms such as HTTP, Internet e-mail, TCP/IP.

Glue is an optional part that glues the network-independent buyer application to a user application such as a WWW browser, a CD-ROM catalog, etc. (see [26])

Payment applications, Transaction Layer, Glue, Comm and Safe Storage are all implemented in C++. The lower layers, consisting of the *i*KP, certificate and crypto libraries, are written in C.

The *i*KP Library [27] provides the core functionality for composing and verifying *i*KP protocol messages, using Certificate Library [28] for verifying certificates and Crypto [29] for accessing cryptographic primitives. The *i*KP Library allows Buyer, Seller and Acquirer applications to verify received *i*KP messages against a context kept by the Transaction Layer, and to compose *i*KP messages to be sent. It returns an updated state to the Transaction Layer after each successful verification or composition.

Crypto separates the *i*KP protocol functionality from the cryptographic functionality, and allows support for different cryptographic toolkits. The Crypto API consists of methods for signature generation and verification, encryption and decryption, hashing and key handling (generation, destruction). The Z*i*P Crypto module is based on the RSA and MD5 functions of the RSA BSAFE 2.1 library. It provides additional functionality such as the randomized and plaintext-aware encryption (using OAEP) described in Appendix A. It also provides an interface for seeding the random number generator and implements seed collection combining various sources of randomness such as network

[7] Since multiple clearance transactions against the same payment authorization are allowed, the clearance signatures (SigClrn_S, SigClrn_A) are further distinguished by CLRNSEQ.

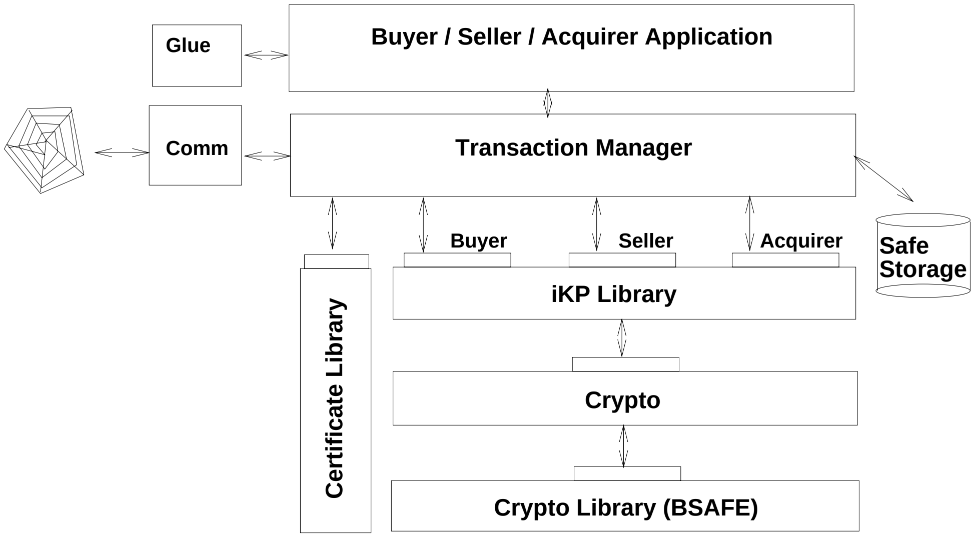


Fig. 9
Z*i*P implementation architecture

traffic and inter-keystroke intervals of user-provided data.

Z*i*P also implements a *dummy* Crypto module which allows for non-export-controlled and platform-independent testing, and serves as an example for implementors of new crypto modules.

The Certificate Library implements a simple certificate issuing and verification functionality for certificates and certificate chains in a hierarchical model. The content of the certificates is taken from the X.509 [30] specification.

For a complete set of Z*i*P documentation, check our web server on `http://www.zurich.ibm.com/Technology/Security/extern/ecommerce/iKP_overview.html`.

H. Deployment

At Europay's Annual Members' Meeting in Seville, Spain, in June 1996, Europay and IBM jointly ran a small-scale trial allowing visitors to the conference to use their pre-loaded Europay CLIP purse card to make secure internet payments from a card reader-equipped terminal. The payment scheme used was an integration of CLIP card payment functionality with Z*i*P-3KP, resulting in a secure scheme for Internet payments from pre-loaded purses.

From April 1997 till February 1998, the EMP (Electronic Market Place) project in Japan, funded by MITI (Ministry of International Trade and Industry), deployed Z*i*P-3KP in a trial with 5 on-line merchants and 2000 users. Each user received a smartcard storing his/her Z*i*P account (keys and certificates), allowing secure Internet purchases from public kiosks and terminals.

Z*i*P is also the payment technology behind the I-Pay payment product offered by Interpay Nederland and the Dutch banks. I-Pay was launched as a trial in June 1996, offering debit-type purchases from twenty on-line shops, using Z*i*P-3KP. Later, Eurocard/Mastercard credit card payments were added to the I-Pay brand. Currently I-Pay is accepted by 80 on-line merchants and has a user base of 17000 users. In line with initial plans to use *i*KP only for the initial trial and to move to the more standardized SET [5] technology once available and accepted, a phased transition is currently replacing Z*i*P technology with SET technology.

Acknowledgements

We thank Phil Janson and Mark Linehan for helpful discussions, and José L. Abad Peiro, Hans Granqvist and Steen Larsen for their contributions to the implementation of *i*KP/Z*i*P.

References

[1] Paul Wallich, "Cyber view: How to steal millions in champ change," *Scientific American*, pp. 32–33, August 1999.

[2] Ross Anderson, "Why cryptosystems fail," *Communications of the ACM*, vol. 37, no. 11, pp. 32–41, Nov. 1994.

[3] Alan O. Freier, Philip Kariton, and Paul C. Kocher, "The SSL protocol: Version 3.0," Tech. Rep., Internet Draft, 1996, Will be eventually replaced by TLS[7].

[4] Stephen Kent and Randall Atkinson, "Security architecture for the Internet Protocol," Internet RFC 2401, Nov. 1998.

[5] Mastercard and Visa, *SET Secure Electronic Transactions Protocol*, version 1.0 edition, May 1997, Book One: Business Specifications, Book Two: Technical Specification, Book Three: Formal Protocol Definition. Available from http://www.setco.org/set_specifications.html.

[6] N. Asokan, Phil Janson, Michael Steiner, and Michael Waidner, "State of the art in electronic payment systems," *IEEE Computer*, vol. 30, no. 9, pp. 28–35, Sept. 1997, A Japanese translation of the article appeared in pp 195-201, *Nikkei Computer* (http://nc.nikkeibp.co.jp/jp/) issue of March 30, 1998.

[7] Tim Dierks and Christopher Allen, "The TLS protocol version 1.0," Internet RFC 2246, Jan. 1999.

[8] William R. Cheswick and Steven M. Bellovin, *Firewalls and Internet Security – Repelling the Wily Hacker*, Professional Computing Series. Addison-Wesley, 1994, ISBN 0-201-63357-4.

[9] Pau-Chen Cheng, Juan Garay, Amir Herzberg, and Hugo Krawczyk, "A security architecture for the internet protocol," *IBM Systems Journal, Special issue on the Internet*, vol. 37, no. 1, pp. 42–60, 1998, Updated version of [31].

[10] N. Asokan, Victor Shoup, and Michael Waidner, "Asynchronous protocols for optimistic fair exchange," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1998, Research in Se-

curity and Privacy, pp. 86–99, IEEE Computer Society Press, A minor bug in the proceedings version was fixed. An errata sheet, distributed at the conference, is available at `http://www.zurich.ibm.com/Technology/Security/publications/1998/ASW98-le\%errata.ps.gz;` A related paper [32] is available as well.

[11] N. Asokan, *Fairness in Electronic Commerce*, Ph.D. thesis, University of Waterloo, May 1998.

[12] Ron Rivest, "The MD5 message-digest algorithm," Internet RFC 1321, Apr. 1992.

[13] NIST National Institute of Standards and Technology (Computer Systems Laboratory), "Secure hash standard," Federal Information Processing Standards Publication FIPS PUB 180-1, Apr. 1995.

[14] Mihir Bellare, Ran Canetti, and Hugo Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology – CRYPTO '96*. 1996, number 1109 in Lecture Notes in Computer Science, pp. 1–15, Springer-Verlag, Berlin Germany.

[15] Hugo Krawczyk, "Blinding of credit card numbers in the SET protocol," in *Proceedings of the 3rd Conference on Financial Cryptography (FC '99)*, Anguilla, British West Indies, Feb 1999, International Financial Cryptography Association (IFCA).

[16] Mihir Bellare and Phillip Rogaway, "Optimal asymmetric encryption – how to encrypt with rsa," in *Advances in Cryptology – EUROCRYPT '94*, I.B. Damgard, Ed. 1994, Lecture Notes in Computer Science, pp. 92–111, Springer-Verlag, Berlin Germany, final (revised) version appeared November 19, 1995.

[17] Ronald Cramer and Victor Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *Advances in Cryptology – CRYPTO '98*, Hugo Krawczyk, Ed. Aug. 1998, number 1462 in Lecture Notes in Computer Science, pp. 13–25, Springer-Verlag, Berlin Germany.

[18] Eui-Suk Chung and Daniel Dardailler, "Joint electronic payment initiative (jepi)," White paper, JEPI, Apr. 1997.

[19] Michael Waidner, "Development of a secure electronic marketplace for Europe," in *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS)*, E. Bertino, H. Kurth, G. Martella, and E. Montolivo, Eds., Rome, Italy, Sept. 1996, number 1146 in Lecture Notes in Computer Science, Springer-Verlag, Berlin Germany, also published in: EDI Forum 9/2 (1996) 98-106, see also `http://www.semper.org`.

[20] *SEMPER* Consortium, "Final report of project *SEMPER*," Deliverable D13 of ACTS project AC026, Aug. 1999, To appear in the LNCS Series, Springer Verlag.

[21] Shai Halevi and Hugo Krawczyk, "Public-key cryptography and password protocols," *ACM Transactions on Information and System Security*, vol. 2, no. 3, pp. 25–60, 1999, Preliminary version in Proc. of the 5th ACM Conference on Computer and Communications Security, 1998, pp. 122-131.

[22] Ralf Hauser, Michael Steiner, and Michael Waidner, "Micropayments based on iKP," Research Report 2791 (# 89269), IBM Research, Feb. 1996.

[23] Steve Glassman, Mark Manasse, Martin Abadi, Paul Gauthier, and Patrick Sobalvarro, "The millicent protocol for inexpensive electronic commerce," in *Fourth International Conference on the World-Wide Web*, MIT, Boston, Dec. 1995.

[24] Benjamin Cox, J. D. Tygar, and Marvin Sirbu, "NetBill security and transaction protocol," In *First USENIX Workshop on Electronic Commerce* [33].

[25] Steen Larsen, *Zurich iKP Prototype (ZiP): iKP Transaction Layer Functional Specification*, IBM Zurich Research Laboratory, May 1996.

[26] Ralf Hauser and Michael Steiner, "Generic extensions of WWW browsers," In *First USENIX Workshop on Electronic Commerce* [33], pp. 147–154.

[27] Gene Tsudik, "Zürich iKP prototype: Protocol specification document," Research Report RZ 2792, IBM Research, Feb. 1996.

[28] Els Van Herreweghen, *Zurich iKP Prototype (ZiP): Certificate Library (CERT) Specification*, IBM Zurich Research Laboratory, Feb. 1996.

[29] Michael Steiner, *Zurich iKP Prototype (ZiP): Cryptographic Library Specification*, IBM Zurich Research Laboratory, Mar. 1996.

[30] ISO/IEC, "Information technology - open systems interconnection - the directory: Authentication framework," June 1994, same as ITU-T Rec X.509.

[31] P. Chen, J. Garay, A. Herzberg, and H. Krawcczyk, "Design and implementation of modular key management protocol and IP Secure Tunnel on AIX," in *Proc. 5th USENIX UNIX Security Syposium*, Salt Lake City, Utah, June 1995.

[32] N. Asokan, Victor Shoup, and Michael Waidner, "Optimistic fair exchange of digital signatures," in *Advances in Cryptology – EUROCRYPT '98*, Kaisa Nyberg, Ed. 1998, number 1403 in Lecture Notes in Computer Science, pp. 591–606, Springer-Verlag, Berlin Germany.

[33] USENIX, *First USENIX Workshop on Electronic Commerce*, New York, July 1995.

Contents

List of Figures

Mihir Bellare received his BS (in Mathematics) from the California Institute of Technology in 1986, and his Ph.D (in Computer Science) from the Massachusetts Institute of Technology in 1991. He worked at IBM before taking a faculty position at the University of California at San Diego. He is a recipient of a David and Lucile Packard Fellowship in Science and Engineering, and a NSF CAREER award. His research areas are cryptography and complexity theory. He is one of the designers of HMAC (an Internet standard for message authentication) and of OAEP (an RSA based encryption scheme currently being proposed to replace RSA PKCS #1). He has published about 40 papers in cryptography (including 26 in the Crypto and Eurocrypt conferences) and about 20 in complexity theory.

Juan A. Garay received his Ph.D. in Computer Science from Penn State University in 1989. He also holds the degree of Electrical Engineer from the Universidad Nacional de Rosario in Argentina, and a Master's in EE from the Netherlands Universities Foundation (PII) in Eindhoven, Holland. He has been with the Secure Systems Research Department of Bell Labs since 1998. From 1990 until 1998 he was with IBM's T.J. Watson Research Center. In 1992 he was a postdoctoral fellow at The Weizmann Institute of Science in Israel, and in 1996 a visiting scientist at the Centrum voor Wiskunde en Informatica (CWI) in Holland. Dr. Garay has published extensively in the areas of algorithms, distributed computing, fault tolerance, and cryptographic protocols.

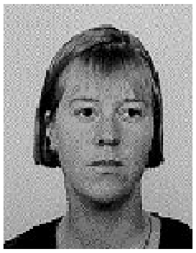
Ralf Hauser holds a M.Sc. in Computer Science from the University of Toronto and a Ph.D. from the University of Zurich. From 1992-1995 he worked as a researcher with the IBM Research Laboratory in Zurich in the field of network security and since, he is a consultant with McKinsey & Co. Since December 1998, he is building for the recently founded McKinsey Business Technology Office a Global Knowledge Management Team (Service and Infrastructure).

Amir Herzberg received the B.Sc. (Computer Engineering), M.Sc. (Electrical Engineering) and D.Sc. (Computer Science) degrees from the Technion – Israel Institute of Technology, in 1982, 1986, and 1991, respectively. Since 1991, he is with the IBM Research Division, currently as manager of the E-Business and Security Department at the Haifa Research Lab. Previously he managed Network Security at the Watson Research Center. He has authored numerous papers and patents. His research areas include electronic commerce, network security, applied cryptography, communication protocols and fault tolerant distributed algorithms.

Hugo Krawczyk is an associate professor at the Department of Electrical Engineering, Technion, Israel, and a visiting scientist at the IBM T.J. Watson Research Center. He received his Ph.D. in Computer Science from the Technion, Israel, in 1990. From 1991 to 1997 he was a research staff member in the Cryptography and Network Security Group at the IBM T.J. Watson Research Center. His areas of interest span applied and theoretical aspects of cryptography with particular emphasis on applications to network security. In particular he has been a co-designer of the HMAC authentication function, the IKE key exchange protocol for IPSEC, and of the SET protocol for secure credit card transactions over the Internet.

Michael Steiner is a research scientist at the Department of Computer Science, Universität des Saarlandes, Saarbrücken and in the network security research group at the IBM Zurich Research Laboratory. His interests include secure and reliable systems as well as cryptography. He received a Diplom in computer science from the Swiss Federal Institute of Technology (ETH) in 1992 and expects to receive a Ph.D. in computer science from the Universität des Saarlandes, Saarbrücken.

Gene Tsudik is a project leader at USC/ISI and a research associate professor in the Computer Science Department at USC. His research interests include network security, applied cryptography and routing in wireless networks. He received a Ph.D. in Computer Science from USC in 1991 and spent the next five years at IBM Research working on secure systems, protocols, mobile networks and electronic commerce. At USC, he teaches courses in Cryptography, Computer Security and Wireless Networks.



Els Van Herreweghen received a degree in Chemical Engineering (Ingenieur Scheikunde en Landbouwindustrieën) and a Master's degree in Computer Science (Licentiaat Informatika) from the Katholieke Universiteit Leuven, Belgium, in 1988 and 1992, respectively. Since 1992, she has been a Research Staff Member in the Network Security group at the IBM Zurich Research Laboratory, Switzerland. Her current research focuses on security issues related to electronic commerce.

Michael Waidner is the manager of the network security research group at the IBM Zurich Research Laboratory. His research interests include cryptography, security, and all aspects of dependability in distributed systems. He has coauthored numerous publications in these fields. Dr. Waidner received his diploma and doctorate in computer science from the University of Karlsruhe, Germany.

Appendix

I. The Encryption Function

*i*KP requires the buyer to encrypt the SLIP as part of the Payment message. (The sole exception to this is when both the PFLAGS:noEnc and the PFLAGS:SIG_B option flags are set.) In this section we describe the implementation of OAEP [16], the encryption function used by *i*KP/Z*i*P.

A. Payload

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:

$$\begin{aligned}\text{SLIP} = \quad & [\,\text{AUTHPRICE}, \mathcal{H}(\text{Common}), \text{BAN}, R_B,\\ & [\text{SALT}_C|\text{PIN}], \text{EXPIRATION}, \text{PADDING}\,]\end{aligned}$$

Fields within SLIP are described in Figure 5. The sizes of these fields are given in Figure 10. All of the above components are encoded into a 896-bit long string.

AUTHPRICE	64 bits
BAN	0-128 bits (128 bits can be used to encode up to 38 decimal digits. Current credit cards are only 12 digits.)
EXPIRATION	32 bits
SALT$_C$ or PIN	0-64 bits (up to 19 decimal digits)
$\mathcal{H}$(Common)	128 bits
R_B	At least 128 bits
PADDING	length is the difference between 832 bits and the sum (in bits) of all previous fields.

Fig. 10

Sizes of fields in EncSlip

B. Encryption process

The actual encryption process is adapted from [16]. The encryption function $\mathcal{E}_A$ is based on RSA. We let $f(x) = x^e \pmod N$ denote the RSA function and $f^{-1}(y) = y^d \pmod N$ its inverse, where N is a 1024 bit modulus. The issue is that simply encrypting under RSA– ie. setting $\mathcal{E}(x) = f(x)$– is not enough: this doesn't provide the "integrity" or "plaintext awareness" we need. Instead, we first "embed" a up to 832-bit plaintext into a 1024 bit string r in a special way and then compute $f(r)$. The scheme we now describe is a simplification of a OAEP scheme from [16]. It makes use, in addition to RSA, of MD5 as the hash function $\mathcal{H}$, and is provably secure assuming $\mathcal{H}$ behaves like a "random function."

The encryption process is illustrated in Figure 11 and performs following steps:

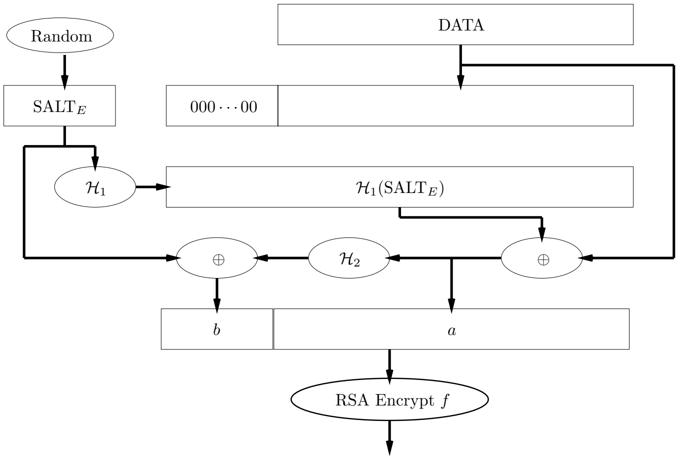


Fig. 11

Encryption using OAEP

1. Prepend 64 bits of zeros to 832 bits of DATA to form $x = [0 \ldots 64x \ldots 0, \text{DATA}]$.
2. Generate random 128-bit string SALT$_E$.
3. Compute $a = x \oplus \mathcal{H}_1(\text{SALT}_E)$.
4. Let $b = \text{SALT}_E \oplus \mathcal{H}_2(a)$.
5. Compute $f(a, b)$ (combined length of a, b is 1024 bits).

$\mathcal{H}_1$ is a one-way function which expands data from one block of 128 bits to 896 bits and is illustrated in Figure 12.

$\mathcal{H}_2$ is a one-way function which compresses data of size 896 to a block of 128 bits. See Figure 13 for its implementation.

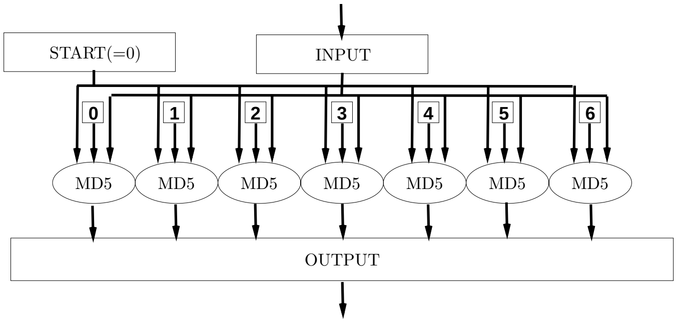


Fig. 12

Hash-function $\mathcal{H}_1$ for OAEP

C. Decryption process

The decryption process is illustrated in Figure 14 and performs the following steps:

1. Compute $(a, b) = f^{-1}(\text{ENCRYPTED}_\text{D}\text{ATA})$.
2. Compute $\text{SALT}_E = b \oplus \mathcal{H}_2(a)$.
3. Compute $x = a \oplus \mathcal{H}_1(\text{SALT}_E)$.
4. Check for existence of 64 leading 0-s in x and obtain DATA.

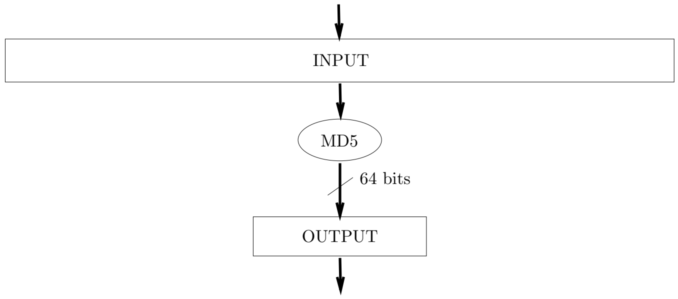


Fig. 13
Hash-function $\mathcal{H}_2$ for OAEP

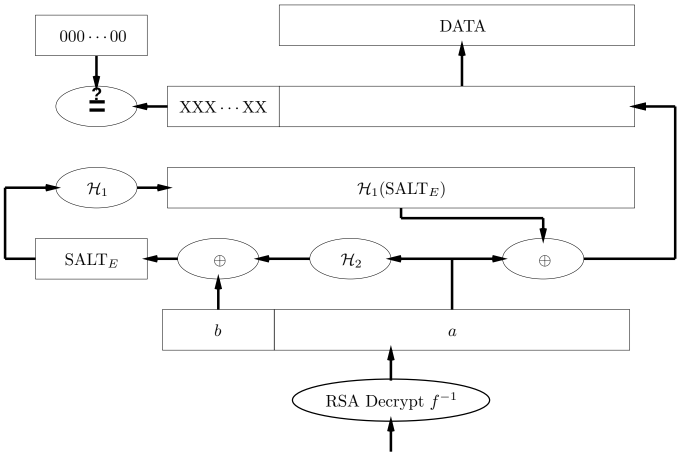


Fig. 14
Decryption using OAEP