Reputation Service for Use in Developing Countries

1 Introduction

We have worked with a company called Ubuntu Capital (`http://ubuntucapital.com`) to design a professional reputation service for use in developing countries. There are a number of security and general integrity considerations for such a system, particularly provided the constraint that certain functions must be available using a basic smartphone without reliable access to the Internet. In response to Ubuntu Capital's business needs, we have focused primarily on designing a solution for one function within this system: the secure offline verification of an individual's reputation, as indicated by a history of economic transactions with other users that have been verified and stored on a server.

This document begins with a brief introduction of Ubuntu Capital and the market need it is serving. This is followed by a discussion of the specific business need we addressed (i.e. verification) and our proposed solution. Brief discussions of adjacent needs and technical functions (e.g. user authentication and creating new transactions) are also included.

1.1 Background

Founded in Cambridge, MA and with operations in Kampala, Uganda and Nairobi, Kenya, Ubuntu Capital is addressing two needs in the region:

1. High search costs when hiring contract service providers, e.g. electricians.
2. Prohibitive diligence costs when evaluating the creditworthiness of applicants for business loans.

Both needs are rooted in the same cause - upwards of 90% of East Africa's economy is driven by the informal sector, i.e. individuals and small businesses completing business transactions amongst one another without being taxed or otherwise monitored by any form of government. While the absence of a paper trail may sound like a pretty good deal, it makes it difficult for skilled service providers to prove their credibility to potential clients and loan officers who are outside their immediate professional and social networks. Their work product is typically not portable nor readily assignable to them and, further, bad actors - individuals who falsely portray themselves as skilled in a domain in order to get a job - are prevalent and the costs they impose can be substantial. For example, consider what could go wrong if an unskilled electrician is hired to run wires through a house. As a result, risk aversion is high among employers, skilled service providers are under-employed, and otherwise ready-to-invest capital remains on the sidelines.

Interestingly, this "trust challenge" is exacerbated by rising urbanization and the creation of transnational special economic zones (SEZs). As individuals have migrated away from their home communities, traditional tribal trust networks have broken down and, further, high levels of itinerancy amongst urban workers has made it difficult for new trust networks to form.

Ubuntu Capital is seeking to overcome these challenges by allowing individual service providers to build their professional reputations using a two-sided platform through which past clients can rate their work and, reciprocally, service providers can rate the client. Thus, when the service provider approaches a new potential client, they can point to the volume and quality of their transaction history as proof of their abilities.

Very generally, Ubuntu Capital's platform has three core business functions (as shown in Figure 1):

1. **Record:** Allow buyers and sellers to record transactions (defined in next section).
2. **Search:** Help buyers search for qualified sellers; help sellers search for posted jobs.
3. **Verify:** Enable buyers to verify the transaction history of sellers.

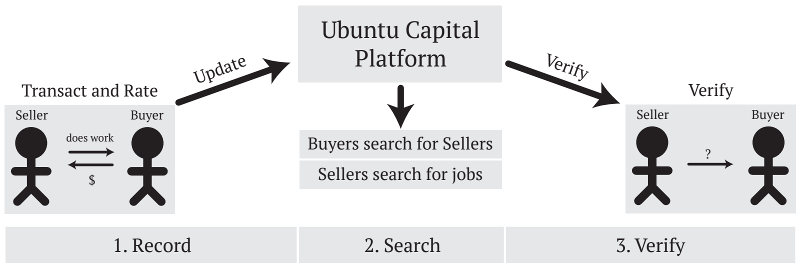

Figure 1: Core functions of Ubuntu Capital's platform

1.2 General security and integrity considerations

Beyond the application of best practices to authentication, authorization, data management, and the like (briefly discussed later), this system has a number of security and general integrity considerations, including:

1. Identity management, i.e. ensuring a 1-to-1 relationship between individuals and accounts.
2. Malicious or overly favorable ratings, i.e. ratings that don't reflect reality.
3. Collusion between individuals, i.e. by creating false transactions and/or ratings.
4. Service delivery given the technical constraints in a developing context, e.g. limited connectivity.

As Ubuntu Capital already has active projects focused on the first three of these considerations (discussed in Section 3), they have asked us to focus on a specific aspect of the last - limited connectivity and its impact on the availability (and perceived reliability) of the service.

While cellular access is reliable in major urban centers, data is expensive and users have voiced that they are hesitant to use *any* data unless absolutely necessary. Surprisingly, this has been cited as a major point of friction in the adoption of the product. Further, wireless access deteriorates rapidly as one moves away from major cities, making the platform's primary value proposition - verifying someone's work history - impossible in the field. These constraints in mind, Ubuntu Capital has asked us how they could:

Enable clients to verify the transaction history of a service provider without an Internet connection.

Thus, while we include a very brief discussion of identity management (Section 3 - "Identity") and the secure generation of new transactions between a buyer and a seller (Section 5 - "Update"), the focus of this document is the secure offline verification of a seller's existing transaction history (Section 4 - "Verification") subject to the assumptions and constraints listed in the next section.

1.3 Business requirements for the verification of a seller's transaction history

This section summarizes the desired user experience and high-level constraints provided by Ubuntu Capital. More detailed, security-focused assumptions and specifications are outlined in subsequent sections.

1.3.1 Desired user experience

As a first step in designing this system, we describe the desired user experience from a purely functional perspective, agnostic of any technology implementation or specific security mechanism. There are two scenarios that must be considered: {Honest Buyer, Honest Seller} and {Honest Buyer, Dishonest Seller}.

The desired experience for an honest buyer (B) and honest seller (S) scenario is as follows:

1. B announces s/he needs a particular service, e.g. an electrician.
2. S approaches B and attests to their skill as an electrician.
3. B requests proof from S.

4. S provides proof.
5. B verifies proof, is satisfied that B is a skilled electrician, and hires B.

The desired experience for an honest buyer (B) and dishonest seller (D) scenario is as follows:

1. B announces s/he needs a particular service, e.g. an electrician.
2. D approaches B and attests to their skill as an electrician.
3. B requests proof from D.
4. D provides a forged proof.
5. B fails to verify proof and does not hire D.

1.3.2 Business constraints and assumptions

Ubuntu Capital has also provided several business assumptions and constraints, the full set of which is included in the next section of this report. Those specifically related to verification include:

1. Proof verification must be possible without Internet access.
2. Each user must be able to verify other users living within a 250 mile radius (assume 500K users).
3. Each user will have Wi-Fi access (and unlimited data) every two weeks.

The next section provides an overview of the system, including how its structured and its key functions, as well as a complete set of assumptions and constraints.

2 System Overview

This section offers an overview of the system, including stakeholders, components, functions, constraints, assumptions and data structures.

2.1 Definitions

The following are a collection of key definitions that will be referenced in later discussions (and are not intended to be a full security policy!).

2.1.1 Stakeholders

1. Ubuntu Capital: the trusted firm that builds and maintains the platform outlined below.
2. Users: individuals* who have been granted access to the platform by Ubuntu Capital.

**Assume 1-to-1 platform user accounts to physical individuals (see "Identify" section)*

2.1.2 Roles

1. Buyer (B): user role, assumed when purchasing a service from a seller.
2. Seller (S): user role, assumed when selling a service to a buyer.

2.1.3 System components

1. Platform: the collection of services (discussed below) provided by Ubuntu Capital via a server.
2. Transaction: discrete record stored by the platform detailing an exchange between a buyer and seller.
3. Table of transactions: the server-side collection of all transactions between buyers and sellers.
4. App: Ubuntu Capital's native mobile application.
5. Seller's (Buyer's) device: the seller's (buyer's) mobile device with accompanying App.

2.1.4 System functions

1. Identify: enable a buyer to link a seller (physically present) to a user account (discussed in Section 3)
2. Verify: enable a buyer to verify a seller's transaction history (discussed in Section 4)
3. Update: enable a buyer and seller to record a new transaction (discussed in Section 5)

2.2 Business constraints and assumptions

As discussed previously, the target market for this reputation service includes individuals transacting within the informal economy in a still-developing context. Cellular connectivity is expensive and unreliable. As a result, the system functions defined above must be possible without a real-time connection to the Internet. The complete set of business-side constraints and assumptions we're designing to are as follows (as originally presented in the previous section):

1. Ubuntu Capital has a server ("Platform") that contains the full list of buyer-seller transactions.
2. Best practices are used for secure server-side data storage and permissions management.
3. Best practices are used for remote client-server authentication, authorization, and data transmission.
4. Each user has a basic smart phone that is password protected and used only by them.
5. Each user has an Ubuntu Capital mobile application ("App") that is also password protected, i.e. security best practices are used for device-side authentication, authorization, and data storage.
6. The App has unrestricted access to local smart phone capabilities, including:
   (a) Local processing (1.2GHz min)
   (b) Local storage (50MB max, encryption possible)
   (c) Camera
7. The App has intermittent access to remote services, including:
   (a) Location (i.e. GPS)
   (b) Uploading data to the platform (via HTTPS)
   (c) Downloading data from the platform (via HTTPS)
8. Verification (or failed verification) must be possible without access to the platform.
9. Each user must be able to verify other users living within a 250 mile radius (assume 500K users).
10. Each user will have Wi-Fi access (and unlimited data) every two weeks.

2.3 Storage requirements

The two stakeholders in the system fall into three categories. Users can be a buyer or buyer/seller while Ubuntu Capital provides and secures a server. Each user at a minimum has a 256 bit signature signing key (ECDSA with NIST P-256 is used as the signing scheme for this paper).

2.3.1 Buyer Requirements

In addition to the user's signature signing key, buyers are also required to keep a single table which contains the following data for each user:

1. User ID (32 bits)
2. Signature verification key (256 bits)
3. SHA256(Seller's Merkle Root || Buyer ID) (256 bits)
4. Timestamp of last update (64 bits)
5. Reputation for the user (16 bits)

Each entry would require 624 bits (excluding overhead); for 500,000 people on the network, this would require ˜39 MB.

2.3.2 Seller Requirements

In addition to the user's signature signing key, the seller is also required to keep two tables, the buyer table and a table of his own transaction blocks, with each block defined as:

1. Transaction ID (64 bits)
2. Timestamp (64 bits)
3. Seller ID (32 bits)
4. Buyer ID (32 bits)
5. σ_{Seller}(SHA256(Buyer ID ∥ Timestamp ∥ Transaction ID)) (512 bits)
6. σ_{Buyer}(SHA256(Seller ID ∥ Timestamp ∥ Transaction ID)) (512 bits)
7. Seller Sync Timestamp (64 bits)

Each block would require 1,280 bits (excluding overhead); for a given seller with 1000 successful transactions this would require 160 kB. The buyer table is necessary primarily for user signature verification keys. Overall this should require a total of 40 MB per seller.

2.3.3 Server Requirements

The server is required to keep the most up-to-date copies of the buyer table as well as a seller's table for all users. This would require 80 gigabytes of storage for a fully loaded network with 1,000 transactions for each of the 500,000 users.

3 Function: Identify

This section very briefly discusses the security considerations associated with the platform's "Identify" function which, in the context of the verification process we're considering, enables a buyer to link a seller (physically present) to a specific user account.

3.1 Overview

As mentioned in the Introduction and System Overview sections, Ubuntu Capital is currently exploring a handful of options to enable the "Identify" (a/k/a authentication) function in its service. Namely, create a way for a Buyer to ensure that the person who is physically presenting him/herself as the Seller associated with a transaction history is, in fact, the physical person associated with that account (and vice versa). While this function is out of scope for this project, it warrants a brief discussion and general synthesis of what we discovered.

The challenge of ensuring the integrity of user accounts and their associated transaction histories has 3 parts:

1. Maintaining a 1:1 relationship between physical people and accounts on the platform.
2. Ensuring transactions were actually completed by the person associated with the linked account.
3. Authenticating a physical person as a user and associated transaction history.

3.2 Maintaining a 1:1 relationship between physical people and accounts

Ubuntu Capital currently requires that a unique SIM card and email account pair be used when creating or modifying a user account. While this does introduce some friction/cost for users who might want to walk away from an unfavorable transaction history or create multiple accounts, it is by no means secure. We explored options in this area very briefly and, surprisingly, found it to be an incredibly difficult problem to solve. Biometrics (e.g., fingerprints, voice analysis, facial recognition) have been used with mixed results and come with the risk that, once one's identity "leaks," it is impossible to simply reset the credentials [8]. Further, particularly in areas such as East Africa, identification credentials issued by central authorities (e.g., regional, national, and supernational government IDs) are viewed with some skepticism after years of largely unchecked forgery. As a result, Ubuntu Capital is now exploring partnerships with 3rd parties that share the same need with the goal of creating a shared service for issuing digital identity credentials based upon a mosaic of factors that, when taken together, ensure a unique identity. For example, a combination of biometrics, state-issued ID(s), in-person "vouching" by others, and local context-aware questioning performed through a network of kiosks.

3.3 Authentication of physical people

Ubuntu Capital currently relies on the Buyer to visually compare a Seller-submitted photo to make sure the person in front of them is actually the user associated with the photo. Provided the photo is an accurate representation of the user, this isn't a completely unreasonable way to perform this check when new transactions are being created or a Seller's transaction history is being verified (this is basically what cab companies and Uber do).

That said, this method creates obvious opportunities for exploitation, particularly when the photo can be changed freely (as is currently the case for Ubuntu Capital) or when photos are unavailable (as will be the case for the offline verification check we explore later in this document). Controlling how and when user photos are uploaded/changed by using, for instance, the kiosk system discussed in Section 3.2 would be an improvement beyond current state. Further, authentication against the digest of a biometric property (e.g. a photo, voice) could be included as part of an offline verification check, as will be discussed in Section 4.

For the offline verification check that is the subject of our design work, we are keeping additional exploration of user identification and authentication out-of-scope. Ubuntu Capital's current assumptions are as follows:

1. Individuals are, by and large, interested in protecting their transaction histories. This includes assigning "good" transactions to themselves and avoiding "bad" transactions that would impair their reputation.
2. Phones and mobile applications are individually password-protected.
3. Accessing an Ubuntu Capital application (and the data therein) satisfies the company's current authentication requirements.

The next section dives into the focus of this document: the offline verification function.

4 Transaction Verification and Reputation Confirmation

In this section, the general flow and various methods for seller verification and reputation confirmation are proposed and discussed. The base case reputation heuristic is explored–a simple count of number of valid transactions performed.

4.1 Goals and Requirements of Transaction Verification and Reputation Confirmation:

The goals of transaction verification and reputation confirmation are:

1. In-person verification of a seller's reputation,
2. Allow a buyer to authenticate a seller, with no dependency on internet connectivity,
3. A buyer cannot use the new information he learned about the seller and act as an adversary. This will prevent both collaboration options between users and conflicts with users who are both buyers and sellers.
4. The authentication process should be fast enough and low in space in reference to low-end smartphone devices.

These goals are achieved through use of a Merkle tree on the seller's device and a hash comparison between the buyer and seller. Overall, this allows for an additional identity confirmation check and ensures that the reputation recorded is correct.

4.2 Transaction Verification and Reputation Confirmation: Base Case

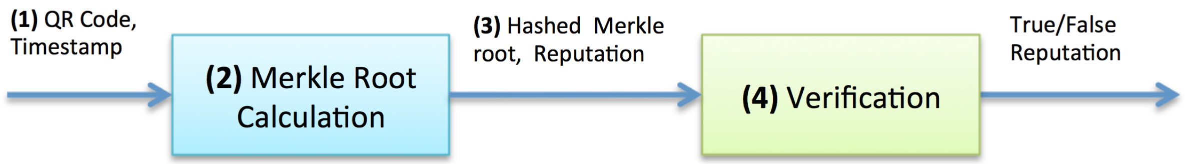
(1) QR Code, Timestamp
(2) Merkle Root Calculation
(3) Hashed Merkle root, Reputation
(4) Verification
True/False Reputation

Figure 2: Block Diagram of Transaction Verification and Reputation Confirmation Process
(Blue represents seller actions, green represents buyer actions)

Transaction verification and reputation confirmation is initiated by the buyer. Initial assumptions is that the seller and buyer have both verified each others identity (as discussed in the previous chapter). As such, the buyer knows who the seller is, knows the seller's hashed SHA256(Seller's Merkle Root || Buyer ID), and knows the rating of the seller based on the look-up table on the buyer's device. The hashed Merkle root is based on the latest time that the buyer has updated his reputation list.

Note that the hash that the buyer has is not the seller's transaction Merkle root but the SHA256(Seller's Merkle Root || Buyer ID). This is because we wish to keep the seller's Merkle root secret and difficult to find; if an adversary was able to obtain the seller's Merkle root, he could theoretically impersonate the seller and provide a valid verification hash.

Following the block diagram found in Figure 2, **(1)** in order for the seller to 'prove' that he is the seller, the buyer provides a QR code scan of his buyer ID and the latest database timestamp:

1. Buyer ID (32 bits): Assigned 32 bit ID for the buyer upon identity registration.
2. Latest Sync Timestamp (64 bit): Timestamp of the last time the buyer synced with the server.

(2) The seller takes all synced transactions prior to the timestamp and calculates the Merkle root of the transactions. The Merkle root is then concatenated with the buyer ID and then hashed, producing SHA256(Seller's Merkle Root || Buyer ID).

The key component of the transaction verification and reputation confirmation is the Merkle Tree root calculation. A transaction leaf in the Merkle tree is defined as the timestamp ordered transaction of the form (discussed in depth in Section 5):

1. Transaction ID (64 bits)
2. Timestamp (64 bits)
3. Seller ID (32 bits)
4. Buyer ID (32 bits)
5. σ_{Seller}(SHA256(Buyer ID || Timestamp || Transaction ID)) (512 bits)
6. σ_{Buyer}(SHA256(Seller ID || Timestamp || Transaction ID)) (512 bits)

For this system, the leaves of the Merkle tree are the seller's individual transactions which are assumed to be secret. The leaves are appended with a 0 bit while the interior nodes appended with a 1 bit in order to reduce the opportunity for a second pre-image attack. Note that at no time does the buyer perform any calculations, have access to Seller Merkle root primitives, or have access to the seller's transactions. If the seller does not leak his transactions, it will be very difficult to produce the correct Merkle root.

(3) The seller then sends back the calculated hash and reputation (number of transactions/leaves of Merkle tree) to the buyer for verification:

1. SHA256(Seller's Merkle Root || Buyer ID) (256 bits)
2. Reputation (16 bits)

(4) If the hashes and reputation align, the buyer is confident that the seller in front of him is the actual seller (in combination with face-to-face identity verification) and that the reputation in the buyer's database is accurate. If not, the reputation of the seller (and the identity of the seller) is suspect as the seller does not have full control of his own transaction chain.

This system does not depend on constant internet connection or an up-to-date database on the buyer's side. As the seller has the list of all transactions for himself, the buyer can at best have the hash of the most current Merkle root. Thus the seller can produce the hashes of all possible Merkle roots that the buyer has (based on the buyer's last sync timestamp), meaning he can prove himself no matter what.

4.3 Calculation/Storage Space Tradeoffs

There is a calculation and storage space tradeoff based on the decision of whether to calculate or store the Merkle root for every verification.

The overview described in the base case currently recalculates the Merkle root for every verification. This option is computation heavy but storage light–only the transactions need to be kept. In this case we will compute a tree for every verification request using the relevant transactions for that timestamp. We will have to hash 2T hashes, where T is the number of transactions in the relevant tree. If a conservative assumption about the processor ISA with respect to the SHA256 implementation of 25 cycles/byte is made, a SHA256 over 256 bits or 32 bytes, will require 800 cycles. With a 1.2 GHz processor, a total of $0.12 * 10^10$ cycles per second can be performed. Thus each SHA256 hash requires 0.667 microsecond of work without additional overhead. Per the worst case, every verification we will have to hash twice as many recorded transactions the user has. Assuming up to 5-second computation time as a upper bound, our system can process around 3.5 million transactions, a number a user will never achieve.

If instead, the Merkle root is stored for every possible transaction and then simply hashed with the buyer ID for verification, verification should be nearly instant. However, this requires storing the key and timestamp for every transaction. With a 256 bit key and 64 bit timestamp over 1000 seller transactions, this would require at maximum 40 kB of space. Additionally, synchronization hashes are computed already for the update process (discussed in Section 5), thus already performing the work necessary to calculate the hash. Based on initial analysis and assuming these assumptions hold, storing the Merkle roots based on timestamps and performing a lookup seem to be the best option.

4.4 Seller Reputation Updates on the Buyer's Device

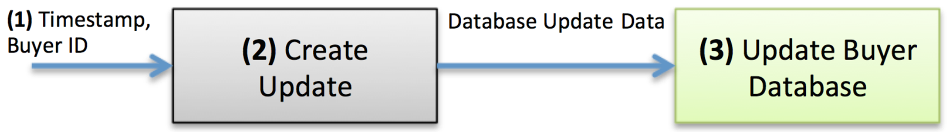

Figure 3: Block Diagram of Buyer's Seller Reputation Table Update
(Grey represents seller actions, green represents buyer actions)

When the buyer is able to obtain internet access, he can request an update of seller reputations in order to match what is currently stored on the server. A database diff of the buyer's table should be used as only two fields need to be updated:

1. SHA256(Seller's Merkle Root || Buyer ID) (256 bits)
2. Reputation for the user (16 bits)

Following the block diagram shown in Figure 3, **(1)** the process will initiate with the buyer sending the server the timestamp and buyer ID of the last time the device was synced. **(2)** After receiving the timestamp, the server will create the relevant table diff data by aggregating only the details of sellers who changed their status since the buyer's timestamp. The SHA256(Seller's Merkle Root ∥ Buyer ID) (256 bits) of the changed entries will be calculated to send to the buyer. The server will then send the diff data consisting of the seller IDs, hashes, and reputations to the buyer who will update his database and timestamp. The timestamp of latest synchronization with the server can be updated for the entire table to reduce bandwidth.

If a diff is used, worst case update size would be 304 bits per person (32 bit seller ID, 256 bit hash, 16 bit reputation) over 500,000 people, or 19 MB. However, this update would only be needed if the every user requires an update which is highly unlikely.

5 Transaction Update

In this section, an outline and various methods for transaction updates are proposed and discussed. Seller and buyer verification of transactions as well as offline transaction generation are of paramount importance.

5.1 Goals and Requirements of Transaction Updating

The goals of transaction updating are as follows:

1. Sellers must be able to create and verify a transaction offline
2. Buyers must be able to verify the same transaction offline
3. (Optional) Completed transactions are rated by the buyer,
4. Sellers must be able to upload new transactions onto the server,
5. Verified transactions must also be verifiable by the server.

These goals and requirements are achieved through the use of digital signatures that are passed back and forth between the seller and buyer.

Note that in this system, transactions are verified by both sellers and buyers. This is different from current reputation services which rely on a one-way review such as Yelp as there is difficulty in determining a valid review from a invalid one. As this system is primarily geared towards in-person transactions with identity verification of both the buyer and the seller, a two-sided transaction verification can take place.

5.2 Seller/Buyer Transaction Update: Base System

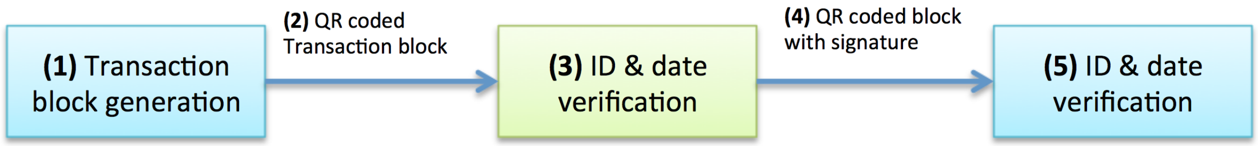
(1) Transaction block generation
(2) QR coded Transaction block
(3) ID & date verification
(4) QR coded block with signature
(5) ID & date verification

Figure 4: Block Diagram of Transaction Update between the Seller and Buyer
(Blue represents seller actions, green represents buyer actions)

In the base system, transaction updating is performed upon completion of the in-person transaction.

Following the block diagram shown in Figure 4, (1) once the in-person transaction is complete, the seller initiates generates a 704 bit transaction block in the following form:

1. Transaction ID (64 bits): Transaction ID is defined as an incremented 2^{32} bit integer concatenated with the seller ID. This is to ensure that the transaction ID is unique even during offline generation.
2. Timestamp (64 bits): Standard UNIX Timestamp representation
3. Seller ID (32 bits): Assigned 32 bit ID for the seller upon identity registration.

4. Buyer ID (32 bits): Assigned 32 bit ID for the buyer upon identity registration.
5. σ_{Seller}(SHA256(Buyer ID ‖ Timestamp ‖ Transaction ID)) (512 bits): Buyer ID is concatenated with the Timestamp and Transaction ID in order to ensure that the plaintext Timestamp and Transaction ID is accurate. This concatenation is then hashed with SHA256 and then finally signed. The size of this signature is 512 bits corresponding to Elliptic Curve Digital Signature Algorithm (ECDSA) using the curve NIST P-256. Technically any hash function can be used as long as the function is collision resistant; similarly any signature scheme can be used–ECDSA was chosen due to its short signature and short keys.

(2) This block is then sent to the buyer in an offline manner. Simple methods of transferring this data between buyer and seller are through QR codes; the expected block is 704 bits which can be encoded in a 37x37 QR code (864 bits at 7% error correction). **(3)** Once the buyer receives the signature, he confirms that his ID and date was hashed and signed correctly by using the seller's public key which is stored locally on the buyer's device. **(4)** Once the buyer's ID is verified, the buyer sends back a similar message:

1. Transaction ID (64 bits)
2. Timestamp (64 bits)
3. Seller ID (32 bits)
4. Buyer ID (32 bits)
5. σ_{Buyer}(SHA256(Seller ID ‖ Timestamp ‖ Transaction ID)) (512 bits): Similar to the previous #5 above, only with buyer and seller actions reversed.

This is sent back to the seller using a QR code; the seller similarly checks for a valid signature. The final transaction block between the buyer and seller is as follows:

1. Transaction ID (64 bits)
2. Timestamp (64 bits)
3. Seller ID (32 bits)
4. Buyer ID (32 bits)
5. σ_{Seller}(SHA256(Buyer ID ‖ Timestamp ‖ Transaction ID)) (512 bits)
6. σ_{Buyer}(SHA256(Seller ID ‖ Timestamp ‖ Transaction ID)) (512 bits)

This transaction block is stored securely on the seller's device and ordered based on Timestamp. Security of these transaction blocks are of highest importance as these transactions will be the basis of the offline verification. Even if multiple transactions are initiated and completed by the seller offline, no transaction ID collision is expected due to the auto-incrementation of the seller's own transaction ID combined with the concatenation of the seller ID.

5.3 Server Transaction Update: Base System

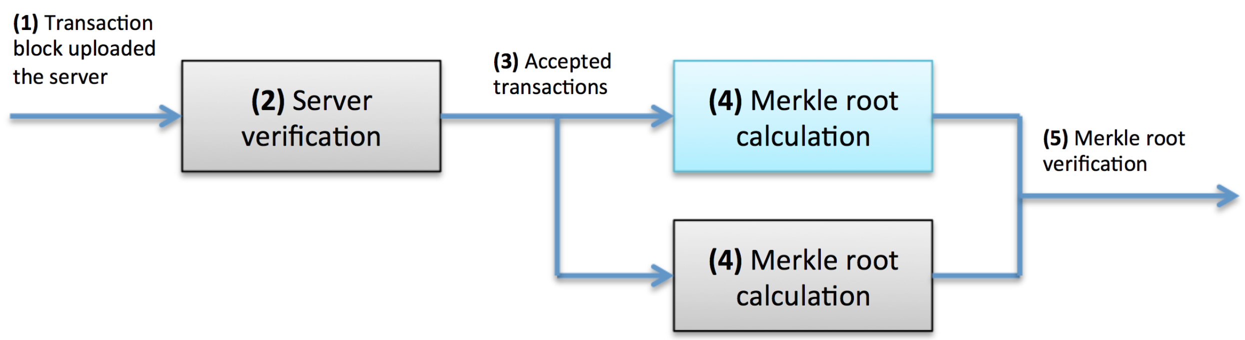
(1) Transaction block uploaded the server
(2) Server verification
(3) Accepted transactions
(4) Merkle root calculation
(4) Merkle root calculation
(5) Merkle root verification

Figure 5: Block Diagram of Server Update between the Seller and Server

(Blue represents seller actions, green represents buyer actions)

In the base system, server updates are performed whenever the seller has access to internet. The goal is to generate the Merkle tree root and update the server such that it contains as many of the seller transactions as possible.

Following the block diagram shown in Figure 5, **(1)** once the seller obtains internet access, transaction blocks since the last sync are uploaded to the server. This data transfer must be performed in a secure manner (e.g. HTTPS) as the transaction blocks are crucial to the verification process; if the transaction blocks are leaked, than verification can be forged (up to the database leak). **(2)** When the transaction block is sent to the server, all the transactions are verified before being appended to the seller's transaction chain. Three checks are performed:

1. Final 32 bits of transaction ID = Seller ID: This check is performed to ensure that the seller ID is correctly identified based on the transaction ID
2. σ_{Seller}(SHA256(Buyer ID $\|$ Timestamp $\|$ Transaction ID)) is a valid seller signature for the buyer ID
3. σ_{Buyer}(SHA256(Seller ID $\|$ Timestamp $\|$ Transaction ID)) is a valid buyer signature for the seller ID

These checks are performed in order to ensure that the seller has not maliciously changed any data within the system.

The first check, the seller ID check, confirms that the transaction ID was generated correctly for the corresponding seller. This however is passed in plain text–an adversary could theoretically change both seller ID and the seller ID portion of the transaction ID.

The 2nd check attempts to neutralize this attack; the second check looks up the seller ID's public key and ensures that the buyer ID, Timestamp, and Transaction ID are correctly signed by the seller ID. If the signature is invalid, the transaction is rejected. As the attacker (hopefully) does not have another seller's signature key, this pairing would be extremely difficult to forge.

Finally the third check primarily ensures that the seller does not generate non-existent transactions; there must be a buyer counter-part to sign the seller ID. Similarly, if this is incorrect, the transaction is rejected. In order for a third party attacker to generate a valid transaction for himself, he must obtain the buyer's signing key; if a third party attacker adjusts a valid transaction, the server should reject the transaction.

(3) Once this process is completed, the server communicates to the seller which transactions were accepted/rejected (transactions can be resent in case of data errors but not explored in this paper). The transactions stored on the seller is thus:

1. Transaction ID (64 bits)
2. Timestamp (64 bits)
3. Seller ID (32 bits)
4. Buyer ID (32 bits)
5. σ_{Seller}(SHA256(Buyer ID $\|$ Timestamp $\|$ Transaction ID)) (512 bits)
6. σ_{Buyer}(SHA256(Seller ID $\|$ Timestamp $\|$ Transaction ID)) (512 bits)
7. Seller Sync Timestamp (64 bits)

(4) When the set of transactions are agreed upon, both the server and seller computes the Merkle tree root with the date-ordered transactions as leaves in the same manner as verification.The final root is concatenated with the seller ID and a final hash computed. **(5)** A final acknowledgement between the server and the seller on the final root is needed to complete the transaction synchronization. Once this is complete, the server updates the seller's reputation to be the number of leaves on the agreed-upon Merkle tree.

5.4 Transaction Updating: Adding Ratings

Implementing a rating structure would allow the server to aggregate ratings for a particular seller and provide this information to a potential buyer in addition to the number of jobs performed. This would allow for better service differentiation for potential buyers.

One way to implement ratings is to establish a rating encoding between the buyer and the server on identification initialization. Essentially, the buyer and the server agree on a rating encoding e.g. 1 star = 1234, 2 stars = 4321, etc. The transaction block is then changed into:

1. Transaction ID (64 bits)
2. Timestamp (64 bits)
3. Seller ID (32 bits)
4. Buyer ID (32 bits)
5. Encoded Rating
6. σ_{Seller}(SHA256(Buyer ID ‖ Timestamp ‖ Transaction ID)) (512 bits)
7. σ_{Buyer}(SHA256(Seller ID ‖ Timestamp ‖ Transaction ID ‖ Encoded Rating)) (512 bits)

The buyer is responsible for providing the encoded rating as well as the new signature; this is then passed back to the seller for upload to the server.

The rating is technically not encrypted; this method is similar to the one-time pad except the pad is being used multiple times and thus loses all cryptography guarantees. In this situation, this sacrifice may be acceptable–the primary purpose of this encoded rating is to make it such that there is some difficulty for the seller to determine what his rating is. As the seller is responsible for uploading the entire transaction to the server, he may reject or delete a transaction if he realizes that his rating is poor. This type of encoding is most insecure in the initial stages of the network; positive and negative ratings changes can be easily seen in the aggregate statistic, thus the rating encoding could be discerned. However, once multiple ratings are aggregated by the server, it will be difficult to determine which rating from which buyer made what impact. Obfuscation can also be employed–ratings can be displayed only in multiples of 5, thus hiding individual effects without a decent body of work.

While encryption of the rating would be best case, the search space for ratings encryption is so small that no real encryption solution was found without an excess burden of key sharing, key rotation, or long ciphertexts.

By implementing this system, the server can calculate a seller's aggregate rating and provide it to the verifier as needed. Once this process is implemented however, the seller has less incentive to solicit ratings if he is unsure about his rating and could bias all ratings on the network.

5.5 Transaction Updating: Adding Project Start/End

Another potential addition to the system could be adding a project start and end checkpoint. This would allow for a calculation of % completed jobs which can be used to determine the trustworthiness of a seller. Project start/end can be implemented by using the same base structure with two more signatures–seller complete, buyer complete. Thus the final transaction block would be:

1. Transaction ID (64 bits)
2. Timestamp (64 bits)
3. Completion Timestamp (64 bits)
4. Seller ID (32 bits)
5. Buyer ID (32 bits)
6. σ_{Seller}(SHA256(Buyer ID ‖ Timestamp ‖ Transaction ID)) (512 bits)
7. σ_{Buyer}(SHA256(Seller ID ‖ Timestamp ‖ Transaction ID)) (512 bits)

8. σ_{Seller}(SHA256(Buyer ID ‖ Completion Timestamp ‖ Transaction ID ‖ "COMPLETE!")) (512 bits)
9. σ_{Buyer}(SHA256(Seller ID ‖ Completion Timestamp ‖ Transaction ID ‖ "COMPLETE!")) (512 bits)

Transaction start would occur the same as the base case; transaction complete would be similar, only a "COMPLETE!" tag is appended to the signature. This would allow for differentiated between initiated but not completed jobs and completed jobs.

This situation follows a similar problem found in the base case–the buyer can withhold closing the transaction until his reasonable or unreasonable demands are completed. The current system in fact provides even more power to the buyer; a 'project start' has been sent to the server (or generated) and cannot be completed without the buyer's consent. Additionally, this system is prone to mistakes–if the seller forgets to "COMPLETE!" the transaction, it may be very difficult to track down the buyer to provide the countersignature.

While this checklist structure may be very useful in judging the reliability of a seller, additional analysis is needed to align the reputation system to actual performance in the field.

6 Security Analysis

In this section, we provide a brief security analysis from the viewpoint of the adversary.

6.1 Identification Vulnerabilities and Potential Solutions

One of the most difficult problems is identification vulnerabilities. While face-to-face to user account verification is not part of the scope of this project, identification is still extremely important to the overall system. While photos tied to user accounts were considered, photos generally take quite a bit of space which may not be feasible for a low-cost phone device.

One idea that was floated for identification verification is in-person challenges. Upon face-to-face meeting, the buyer and seller exchange challenge text which they are required to sign. If the signature is successful, then this is another way to verify the individual. This of course requires that the signing key is not compromised; if individual collude by disseminating signing keys among a small group, the current system cannot be enforced.

6.2 Verification and Update Vulnerabilities and Potential Solutions

Verification vulnerabilities primarily hinge around the transaction blocks. If a seller's transaction blocks are compromised, an attacker will no problems impersonating the seller (up until the signing key is needed). A seller may be able to give away his transaction blocks which can then be aggregated to find the correct Merkle root.

Similarly, if the Merkle tree root hashes are compromised, an attacker will be able to verify as a seller for that point in time. It is due to this vulnerability that each Merkle root is always hashed with a user ID; buyer ID for verification, and seller ID for update. This prevents the transmittance or buyer storage of the transaction Merkle root. Only the seller's phone and the server ever calculate the true transaction Merkle root.

However, even if the transaction list or Merkle root is leaked at a certain point in time, the system may still be recoverable. If the security hole that lead to the transaction list is closed and no further transactions/Merkle roots are leaked in the future, Ubuntu can simply state that verifications prior to the leak date are not valid. As long as the signature signing keys remain secure on the user's device, future transactions can still be added to the seller's reputation chain with no history lost. For example, if on day 10 the database of transactions/Merkle roots are leaked but the security hole is closed, Ubuntu Capital can state that verifications for sellers prior to day 10 are invalid, and only day 11+ verifications should be used.

6.3 Update Vulnerabilities and Potential Solutions

The current primary update vulnerability is seen in the server/seller calculation of the Merkle tree. The leaves only consist of the transactions–theoretically the buyer has a copy of the transaction as well. Once a seller accumulates multiple transactions, multiple buyers who interacted with the seller could collude in order to get all of the transactions of the seller and produce the correct Merkle tree. There are two ways to minimize the impact of this vulnerability: first, during account initiation, a dummy transaction is generated by Ubuntu Capital; as long as Ubuntu Capital is not compromised, at least one leaf in the Merkle tree is unknown, making the Merkle root difficult to calculate. Another way to fix this issue is to generate a random number as part of the transaction that the buyer never sees. Thus, the transaction + this random number is necessary to hash to the correct root. Both of these solutions make it so that it is impossible for all the leaves in the Merkle root to be known for calculation.

6.4 Security Vulnerability Summary

Overall, three key components need to be as secure as possible: seller's transactions, user's signing keys, and server contents. If these three key components can be kept secure, the system should work as intended.

Based on initial analysis, most of the vulnerabilities of this system are based primarily on real-life interactions. The current structure does not preclude real-life shenanigans such as acquaintances colluding to generate legitimate-looking transactions even though the transactions did not actually occur in real-life and seller groups passing around their transactions and signing keys to build up reputation faster. Additionally there is a power imbalance in the current system–the buyer can hold the transaction verification hostage unless the seller performs all work to his satisfaction.

These types of in-person system attacks are not investigated in this paper. Overall however, this structure provides a two-sided transaction block update for use in reputation verification.

7 Conclusion

While there are a number of additional areas to explore for the platform as a whole, our suggested design addresses the (current) greatest point of friction for continued adoption in a way that is sensitive to market context, user need, and security considerations. As with any implementation, our proposal is imperfect, and requires trading robust, in-person authentication capabilities for greater service availability, as driven by the reality of intermittent connectivity in the field. While a multi-factor authentication method, such as one integrating voice or face recognition may mitigate this exposure (with the downsides discussed in Section 3), truly robust in-person authentication remains a challenge even in developed contexts where connectivity is not as great a concern. It will be exciting to continue to work with Ubuntu Capital over the next several months to further the development of a service that, when fully implemented, has the potential to affect so many and on such a large scale.

8 Bibliography

[1] Carboni, D. (2014). Feedback based reputation on top of the bitcoin blockchain. Pua, IT: Center for advanced studies, research, and development in Sardinia.

[2] Christina Garman, M. G. (2013). Decentralized Anonymous Credentials. Baltimore: The Johns Hopkins University Department of Computer Science.

[3] Emmanuel, A. (2007). Mobile Banking in Developing Countries: Secure Framework for Delivery of SMS-banking Services. Radboud University Nijmegen.

[4] G. Zyskind, O. N. (2015). Decentralizing Privacy: Using Blockchain to Protect Personal Data. Cambridge.

[5] ITU. (2006). Cybersecurity guidelines for developing countries. International Telecommunication Union.

[6] R. Dayarathna, H. E. (2003). User Friendly Authentication Mechanism for Rural Communities. Stockholm University.

[7] Vyshegorodtsev, M. (2013). Reputation Scoring System Using an Economic Trust Model: A Distributed Approach to Evaluate Trusted Third Parties on the Internet. Advanced Information Networking and Applications Workshops (WAINA) , 730 - 737.

[8] `https://medium.com/@ConsenSys/tell-me-who-you-are-258268bf3180#.5uphsg9j3`, accessed May 1, 2016.