Lock-In to the Meta Cloud with Attribute Based Encryption With Outsourced Decryption

V.Sudarshan, Dr. N.Satyarayana, A.Dileep Kumar

1. Introduction

The cloud computing paradigm has achieved widespread adoption in recent years. Its success is due largely to customers’ ability to use services on demand with a pay-as-you-go pricing model, which has proved convenient in many respects. Low costs and high flexibility make migrating to the cloud compelling. Despite its obvious advantages, however, many companies hesitate to “move to the cloud,” mainly because of concerns related to service availability, data lock-in, and legal uncertainties. Lockin is particularly problematic. For one thing, even though public cloud availability is generally high, outages still occur. Businesses locked into such a cloud are essentially at a standstill until the cloud is back online. Most public cloud providers’ terms of service let that provider unilaterally change pricing at any time. Hence, a business locked into a cloud has no mid- or longer-term control over its own IT costs. At the core of all these problems, we can identify a need for businesses to permanently monitor the cloud they’re using and be able to rapidly “change horses”—that is, migrate to a different cloud if they discover problems or if their estimates predict future issues. Many companies not (only) build on public clouds for their cloud computing needs, but combine public offerings with their own private clouds, leading to so-called hybrid cloud setups. Here, we introduce the concept of a meta cloud that incorporates design time and runtime components. This meta cloud would abstract away from existing offerings’ technical incompatibilities, thus mitigating vendor lock-in. It helps users find the right set of cloud services for a particular use case and supports an application’s initial deployment and runtime migration.

II. The Meta Cloud

To some extent, we can realize the meta cloud based on a combination of existing tools and concepts, part of which we just examined. Figure 1 depicts the meta cloud’s main components. We can categorize these components based on whether they’re important mainly for cloud software engineers during development time or whether they perform tasks during runtime. We illustrate their interplay using the sports betting portal example.

A. Meta Cloud API

The meta cloud API provides a unified programming interface to abstract from the differences among provider API implementations. For customers, using this API prevents their application from being hard-wired to a specific cloud service. The meta cloud API can build on available cloud provider abstraction APIs, as previously mentioned. Although these deal mostly with key value stores and compute services, in principle, all services can be covered that are abstract enough for more than one provider to offer and whose specific APIs don’t differ too much, conceptually.

B. Meta Cloud Proxy

The meta cloud provides proxy objects, which are deployed with the application and run on the provisioned cloud resources. They serve as mediators between the application and the cloud provider. These proxies expose the meta cloud API to the application, transform application requests into cloud-provider-specific requests, and forward them to the respective cloud services. Proxies provide a way to execute deployment and migration recipes triggered by the meta cloud’s provisioning strategy.
A. New Model of CP-ABE With Outsourced Decryption

In the original model defined in, a CP-ABE scheme with outsourced decryption consists of five algorithms SetUp, Encrypt, KeyGen, Transform, Decrypt. A trusted party uses the algorithm to generate the public parameters and a master secret key, and uses KeyGen to generate a private key and a transformation key for a user. Taking as input the transformation key given by a user and a ciphertext, the cloud can use the algorithm Transform to transform the ciphertext into a simple ciphertext if the user’s attribute satisfies the access structure associated with the ciphertext; then the user uses the algorithm Decrypt, to recover the plaintext from the transformed ciphertext. The input to the algorithm Decrypt includes only the private key of the user and the transformed ciphertext, but does not include the original ciphertext. Because of this omission of the original ciphertext, it is not possible to construct a CP-ABE scheme with verifiable outsourced decryption under the definition of. This can be explained as follows. A malicious cloud could replace the ciphertext it supposes to transform with a ciphertext of a different message, and then transform the latter into a simple ciphertext if the user’s attribute satisfies the access structure associated with the ciphertext. Therefore, the cloud can easily transform a ciphertext CT of a different message into a ciphertext CT’ that satisfies the decryption structure associated with the ciphertext.

IV. Pack Secure Algorithm

The stream of data received at the PACK receiver is parsed to a sequence of variable-size, content-based signed chunks similar to the chunks are then compared to the receiver local storage, termed chunk store. If a matching chunk is found in the local chunk store, the receiver retrieves the sequence of subsequent chunks, referred to as a chain, by traversing the sequence of LRU chunk pointers that are included in the chunks’ metadata. Using the constructed chain, the receiver sends a prediction to the sender.
for the subsequent data. Part of each chunk’s prediction, termed a hint, is an easy-to-compute function with a small-enough false-positive value, such as the value of the last byte in the predicted data or a byte-wide XOR checksum of all or selected bytes. The prediction sent by the receiver includes the range of the predicted data, the hint, and the signature of the chunk. The sender identifies the predicted range in its buffered data and verifies the hint for that range. If the result matches the received hint, it continues to perform the more computationally intensive SHA-1 signature operation. Upon a signature match, the sender sends a confirmation message to the receiver, enabling it to copy the matched data from its local storage.

A. Receiver Chunk Store
PACK uses a new chains scheme, described in Fig. 1, in which chunks are linked to other chunks according to their ast received order. The PACK receiver maintains a chunk store, which is a large size cache of chunks and their associated metadata. Chunk’s metadata includes the chunk’s signature and a (single) pointer to the successive chunk in the last received stream containing this chunk.

Caching and indexing techniques are employed to efficiently maintain and retrieve the stored chunks, their signatures, and the chains formed by traversing the chunk pointers. When the new data are received and parsed to chunks, the receiver computes each chunk’s signature using SHA-1. At this point, the chunk and its signature are added to the chunk store, and the metadata of the previously received chunk in the same stream is updated to point to the current chunk. The unsynchronized nature of PACK allows the receiver to map existing file in the local file system to a chain of chunks, saving in the chunk store only the metadata associated with the chunks. Using the latter observation, the receiver can also share chunks with peer clients within the same local network utilizing a simple map of network drives. The utilization of a small chunk size presents better redundancy elimination when data modifications are fine-grained, such as sporadic changes in an HTML page. On the other hand, the use of smaller chunks increases the storage index size, memory usage, and magnetic disk seeks. It also increases the transmission overhead of the virtual data exchanged between the client and the server.

B. Receiver Algorithm
Upon the arrival of new data, the receiver computes the respective signature for each chunk and looks for a match in its local chunk store. If the chunk’s signature is found, the receiver determines whether it is part of a formerly received chain, using the chunks’ metadata. If affirmative, the receiver sends a prediction to the sender for several next expected chain chunks. The prediction carries a starting point in the byte stream (i.e., offset) and the identity of several subsequent chunks (PRED command). Upon a successful prediction, the sender responds with a PRED-ACK confirmation message. Once the PRED-ACK message is received and processed, the receiver copies the corresponding data from the chunk store to its TCP input buffers, placing it according to the corresponding sequence numbers. At this point, the receiver sends a normal TCP ACK with the next expected TCP sequence number. In case the prediction is false, or one or more predicted chunks are already sent, the sender continues with normal operation, e.g., sending the raw data, without sending a PRED-ACK message.

C. Sender Algorithm
When a sender receives a PRED message from the receiver, it tries to match the received predictions to its buffered (yet to be sent) data. For each prediction, the sender determines the corresponding TCP sequence range and verifies the hint. Upon a hint match, the sender calculates the more computationally intensive SHA-1 signature for the predicted data range and compares the result to the signature received in the PRED message. Note that in case the hint does not match, a computationally expansive operation is saved. If the two SHA-1 signatures match, the sender can safely assume that the receiver’s prediction is correct. In this case, it replaces the corresponding outgoing buffered data with a PRED-ACK message.

V. Conclusion
In this paper, we considered a new requirement of ABE with outsourced decryption: verifiability. We modified the original model of ABE with outsourced decryption proposed. We also proposed a concrete ABE scheme with verifiable outsourced decryption and proved that it is secure and verifiable. Our scheme does not rely on random oracles. To assess the practicability of our scheme, we implemented it and conducted experiments in a simulated outsourcing environment. As expected, the scheme substantially reduced the computation time required for resource-limited devices to recover plaintexts.

References


V. Sudharshan received the M.Tech degrees in computer science Engineering and Information Technology from the JNTUH University, Hyderabad, in 2005 and 2010, respectively, and is currently pursuing the Ph.D. degree in Computer Science & Engineering at the JNTUH—Hyd. He has a vast experience in network-related software-development from several startup companies. His research interests include Secure Cloud Computing, content caching, cellular networks, and traffic redundancy elimination.

Dr. N. Satyanarayana, M.Tech (CS), Ph.D (CSE), The Nagole college is headed by Dr. N. Satyanarayana who possessed the highest qualification in engineering. His qualifications are M.Sc, M.Phil. AMIE (ET), M.Tech (CS), Ph.D (CSE), MISTE, MCSI. He obtained his Ph.D in Computer Science and had a remarkable record of merit in pursuit his engineering studies. He is young and dynamic. He has a disposition to inspire both students and staff to achieve their right goals. With his extensive and rich research experience he is able to run the institution on sound academic ground.

A. Dileep Kumar received the M.Tech degrees in computer science & Engineering from the JNTUH University, Hyderabad, India.