# DXchange: A Digital Property System For People

admin@DXchange.org
www.DXchange.org

**Abstract**. DXchange empowers software developers to mint licenses for their applications which may be traded like non-fungible crypto currency tokens. This system expands the universe of possibilities in software development and delivery.

## 1. Definitions

**DXchange Token (Token)** – A software license which may be traded like a non-fungible crypto currency token.

**DRM –** Digital rights management is system to facilitate the ownership and trading of DXchange tokens.

**DRM Server –** A server program, accessible via the internet, which administers access to software via tokens.

**Software Application** – An executable file with machine instructions for the computer and operating system on which it runs.

**Protected Code** – Encrypted binary code which, once decrypted, must be converted to machine instructions for the computer and operating system on which the Software Application runs.

**Instance Hash** – Binary data maintained in synchronization with a DRM Server by a Software Application.

**Private Key** – Binary data unique to each token, accessible by the Software Application for each token, but not by the DRM Server.

## 2. Introduction

As the value of software applications increases, traditional software companies move to lock down and control application use via centralized systems which prevent users from enjoying ownership rights. Corporations exercise control about who may have access to their software and about which currency systems may be used to buy access. This has prevented innovation and created barriers to entry, both in the development and distribution of new software applications and in the payment systems used to purchase access.

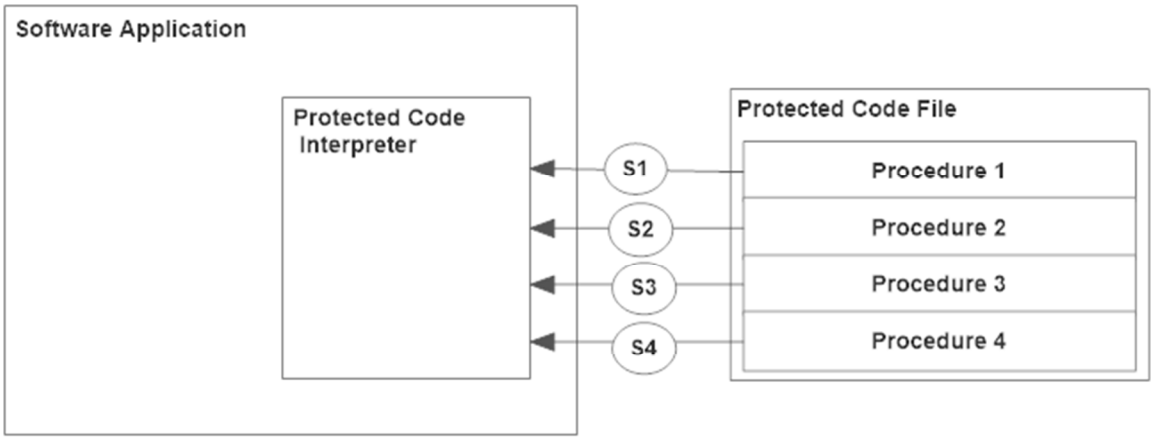
A digital rights management (DRM) system based on cryptographic proof and protected code can allow individuals to transfer ownership of a token without the knowledge or approval of the software application creator.

Such a system facilitates a free market for software application tokens. This free market attracts capital to the world of software development and distribution. The capital will be shared between users and developers and will provide incentives for the creation of new software applications and services that are currently not financially viable.

## 3. Protected Code

Protected code runs within a software application and its behavior may not be modified without causing the application to crash. Protected code prevents hackers from modifying interactions between the software application and a DRM server. The following explains how the DXchange implementation of protected code works.

The software application contains a protected code interpreter which reads and decrypts procedures from an external file. Each procedure in the protected code file has been encrypted with a seed value (S1, S2 … ) designated at the time the protected code file was created. The protected code interpreter attempts to reconstruct the appropriate seed value using the current state of one or more variables when each procedure is invoked. If the state of the protected code has been modified such that the seed value is incorrectly constructed, the procedure will be improperly decrypted and the application will crash.



## 4. Secure Communication and Identity Verification

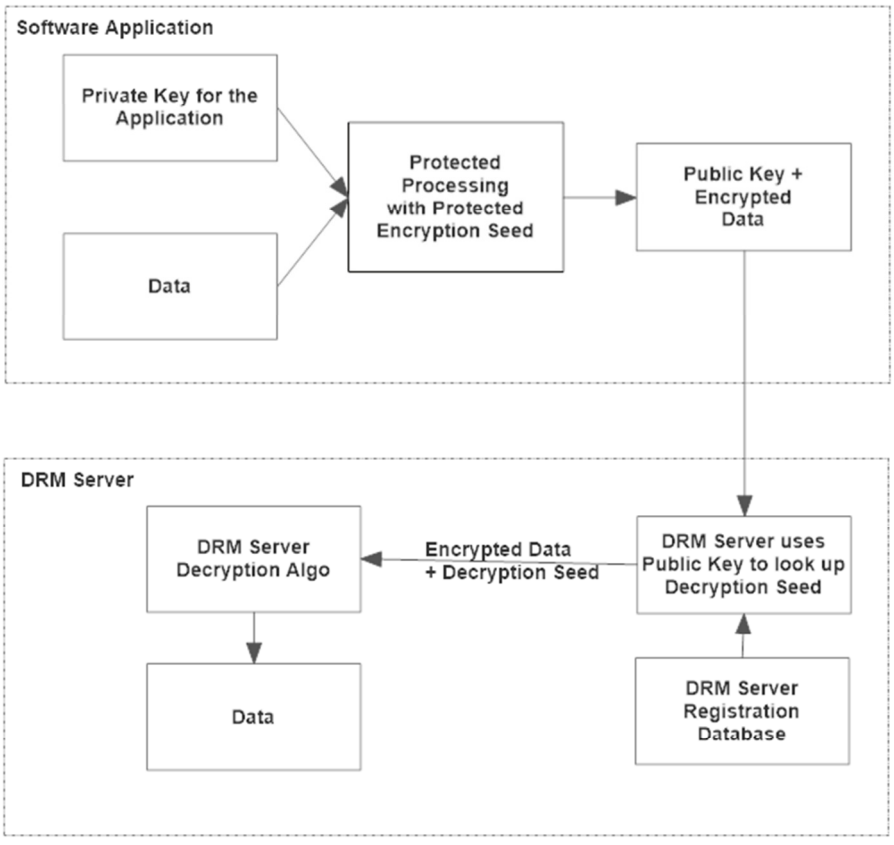
The DRM server maintains a record for every active token. Each record contains 1) a hash of the private key for the token 2) unique random values which are present in the protected code for each token and 3) the instance hash. Note that the DRM server does not have the private key for any token.

Each software application with an active token must include a valid Config.bin file. This is a password-encrypted file containing a private key which uniquely identifies the token, the URL of the DRM server and the instance hash.

The software application symmetrically encrypts data via protected code using algorithms and unique (per token) data embedded in protected code. The DRM server replicates this encryption process using the data saved in the appropriate record.

In order for the software application to accomplish this task, it must have access to the private key, which is hashed as part of the procedure. However, the DRM server can accomplish the same task using only a hash of the private key and data stored in the appropriate record.

Therefore the application software can securely communicate with the DRM server while at the same time proving that it has access to the private key associated with the token.



## 5. Synchronization of Application State with the DRM Server

Synchronization between the software application and the DRM server is required to prevent more than one active instance of a token. Without synchronization, it would be possible to simply copy the Config.bin file, the protected code file and the software application to a new directory or computer and run the software application from that new location as well as from the old location.

To prevent "double spending" of tokens, the software application must verify that its copy of the instance hash, from the Config.bin file, matches the DRM server's instance hash for the same token. Via protected code the software application 1) extracts the instance hash from the Config.bin file 2) sends the instance hash to the DRM server 3) receives back from the DRM server a new instance hash 4) saves the new instance hash to the Config.bin file. If the DRM server fails to provide a valid response in step 3), the protected code enters an invalid state and the software application crashes.

## 6. Decentralization, the Private Key

A fundamental tenet of decentralization is that the end user alone has chain of custody for any private key which defines ownership of property. With this in mind the DRM server was designed so that it does not need the private key.

In a system where the software application generates its own private key, the DRM server must limit the number of tokens which may be added to its registration database.

We can easily envision a system where the software application creates its own private key and where upon registration the DRM server facilitates payment in crypto currency before any token is delivered to the software application.

## 7. Decentralization, the DRM server

In a fully decentralized system, the rules enforced by the DRM server must be unalterable and the DRM server functionality itself must be decentralized in a way which guarantees that it is always available. At this time, a blockchain based system is the only proven way to provide both decentralization and availability.

## 8. Current Implementation

The current implementation of the system includes a program which is not part of the DRM server or the software application, TransctionAgentManager, which generates private keys and protected code for each token. TransactionAgentManager sends the encrypted Config.bin and Data.bin (protected code file) for each token to the DRM

server. The server does not have access to the password of the Config.bin file so does not have access to the private key.

When a user launches an instance of the software application which does not have a Config.bin file associated with it, they are prompted for an integration id. The integration id for each token is created by TransactionAgentManager and may be sent to a user who has paid for a token. The integration id is a single encrypted string containing the URL for the DRM server, the private key for the token, the password for the Config.bin file and an ID which may be used to communicate with the DRM server in order to request download of the Config.bin and Data.bin files currently stored on the DRM server. After the user enters the integration id, the application software interacts with the DRM server to download the Config.bin and Data.bin file contents and once in place, the instance hash is immediately incremented, thereby locking out anyone else who might have the same integration id.

The tasks of maintaining the DRM server and creating new tokens are separated, so incentives to accomplish these two activities may be separated. The owner of the software application's source code will want control over the issuance of new tokens, but may not want to be burdened with the task of managing the DRM server.

## 9. Final Note

The protected code compiler and protected code interpreter used by the system is still evolving; a great deal of work by many people in zero knowledge proofs and secure computing continues. The application of protected code in a DRM system is DXchange's primary accomplishment, a significant step toward empowering individuals and small organizations.