<https://en.bitcoin.it/wiki/Contracts>

（2014年2月25日時点）

# 契約

**分散型契約** とはビットコインを用いてブロックチェーン経由で合意に達するための手法を指します。契約によってこれまで不可能だったことが可能になるわけではありません。むしろ、普遍的な問題を解決する際に、信用を得る必要性を限りなく小さくするためのものです。信用の必要性を最小限に抑えることで、人間の判断が不要となって、より便利になり、完全な自動化が可能になります。

信用が不要なプロトコルを作成してビットコインと連携させることで、これまでとは全く異なる商品を創り出せるようになります。

* [スマート資産](https://en.bitcoin.it/wiki/Smart_Property) はブロックチェーン経由で細かく取引したり、貸与できる資産を指します。
* [転送可能な仮想資産](https://en.bitcoin.it/wiki/Transferable_virtual_property) は、転送可能ですが複製不可能なデジタルアイテムです。
* [エージェント](https://en.bitcoin.it/wiki/Agents) はサーバーの時間を買う目的で自身の財布を管理する自律型プログラムです。エージェントはサービスを売り、収入を得ます。需要が供給を上回る場合、エージェントは子プログラムを生成できます。子プログラムは、注文を得られれば生き残り、得られなければ死滅します。
* [分散型市場](https://en.bitcoin.it/wiki/Distributed_markets) はピアツーピア型の債権と株式取引を実装する手法です。これにより、ビットコインは国際金融システムに完全に競合できるようになります。
* [Rリップル通貨取引所](https://en.bitcoin.it/wiki/Ripple_currency_exchange) はソーシャルネットワークに基づいて分散型の通貨取引所を実装する手法です。

このページにはいくつか実例も記載されています。

Mビットコイン契約の基本となる思想の多くは、Nick Szabóが彼の有名な白書、 [公共ネットワークにおいて関係を構築して強化する方法](http://szabo.best.vwh.net/formalize.html)にて初めて記述されました。以下のページは [Mike Hearn](mailto:mike@plan99.net)によって記載されています。新型の契約のアイデアがある方は、彼に連絡してください。ロンドンで開催されたBitcoin 2012にて発表された [**契約についての講演の映像**](https://www.youtube.com/watch?feature=player_embedded&v=mD4L7xDNCmA) を閲覧できます。

|  |
| --- |
| 内容  [[隠す](https://en.bitcoin.it/wiki/Contracts)]   * [1 Mempool（メモリープール）の取引入替え機能についての警告](https://en.bitcoin.it/wiki/Contracts#A_warning_about_the_mempool_transaction_replacement_mechanism) * [2 理論](https://en.bitcoin.it/wiki/Contracts#Theory) * [3 例 1:預金の入力：](https://en.bitcoin.it/wiki/Contracts#Example_1:_Providing_a_deposit) * [4 例 2:エスクローと仲裁](https://en.bitcoin.it/wiki/Contracts#Example_2:_Escrow_and_dispute_mediation) * [5 例 3:保証契約](https://en.bitcoin.it/wiki/Contracts#Example_3:_Assurance_contracts) * [6 例 4:外部条件を利用する](https://en.bitcoin.it/wiki/Contracts#Example_4:_Using_external_state)   + [6.1 信用を不要にする：課題](https://en.bitcoin.it/wiki/Contracts#Trust_minimization:_challenges)   + [6.2 信用を不要にする：複数の独立したオラクル](https://en.bitcoin.it/wiki/Contracts#Trust_minimization:_multiple_independent_oracles)   + [6.3 信用を不要にする：信頼性の高いハードウェア](https://en.bitcoin.it/wiki/Contracts#Trust_minimization:_trusted_hardware)   + [6.4 信用を不要にする：アマゾンAWSオラクル](https://en.bitcoin.it/wiki/Contracts#Trust_minimization:_Amazon_AWS_oracles) * [7 例 5:チェーンを超えて取引を行う場合](https://en.bitcoin.it/wiki/Contracts#Example_5:_Trading_across_chains) * [8 例 6:検証後の支払：純粋な関数であれば、解決可能です](https://en.bitcoin.it/wiki/Contracts#Example_6:_Pay-for-proof_contracts:_buying_a_solution_to_any_pure_function) * [9 例 7:あらかじめ指定された相手に対して（マイクロ）決済の額を細かく調整](https://en.bitcoin.it/wiki/Contracts#Example_7:_Rapidly-adjusted_.28micro.29payments_to_a_pre-determined_party) * [10 例 8:複数のグループが参加する分散型賭博](https://en.bitcoin.it/wiki/Contracts#Example_8:_Multi-party_decentralised_lotteries) * [11 その他](https://en.bitcoin.it/wiki/Contracts#See_Also) |

## Mempool（メモリープール）の取引入替え機能についての警告

本ページでは、取引のメモリープール入替え用にnSequence を利用できる、と言及しています。この機能は [2010](https://github.com/bitcoin/bitcoin/commit/401926283a200994ecd7df8eae8ced8e0b067c46)年に無効にされ、つい最近になって、コードは [完全に削除されました。](https://github.com/bitcoin/bitcoin/commit/98c7c8fd1d3712e02be0e9f2eeca7e02aa54d197)サービス妨害に用いる人が出るかもしれない、という懸念が原因です。実装を試みる人々はこの点を考慮に入れて、メモリープールの入替えに頼らない契約の仕組みを構築してください。さもなければ、現在のビットコインでは実装結果が動作しません。ビットコインが将来においてメモリープールの入れ替えを再度実装すれば、このページを更新します。

## 理論

ビットコインにおける全ての [取引](https://en.bitcoin.it/wiki/Transaction) は1つまたはそれ以上の入力と出力を持っています。各入力／出力には [スクリプト](https://en.bitcoin.it/wiki/Script)と呼ばれる小型の純粋な機能が関連付けられています。スクリプト内には取引自身の簡素型に対する署名を含めることができます。

全ての取引には、ロック日時が割当てられています。これにより、取引は合意に基づいて未来のある時間までは保留や取替えが可能になります。取引はブロックインデックスまたはタイムスタンプの形で指定されます（同じ内容をどちらの名前で呼ぶこともできますが、5億未満の値はブロックインデックスとして認識されます）。取引がロック日時に達すると、確定と見なされます。

各取引入力には連番を付けることができます。値を動かすだけの、通常取引の場合は、連番は全てUNIT\_MAXと定義され、ロック日時はゼロです。ロック日時に達していなくても、連番が全てUNIT\_MAXであれば、取引は確定した、と見なされます。連番を用いると、署名済み入力をそのまま利用したまま、取引の新バージョンを発行できます。つまり、取引の各入力が別々の人物から送られてくる場合、各入力の番号はゼロから始まりますが、番号は個別に更新できます。

署名の最後に付属するSIGHASHフラグを利用して取引署名を管理できるため、署名チェックも柔軟に行えます。この方法で契約を作成すると、各当事者は自分の領域のみに署名して、他の領域は他の当事者たちに任せることができます。SIGHASHは2つの要素で構成されています：モードとANYONECANPAY変更子です。

1. SIGHASH\_ALL:これがデフォルトです。入力スクリプトを除いて、取引の全てが署名されていることを示します。入力スクリプトも署名してしまうと、当然のごとく取引を構築できなくなってしまうために、入力スクリプトは常に空白になっています。しかし、出力先や連番などの他の入力要素は *署名されている* 点に注意してください。署名されていないのはスクリプト部分だけです。直感的にわかりやすい言い方では、「もし参加者全員が自らのお金を投資して、しかも結果の金額が指定どおりであるならば、自分は投資してもかまわない」となります。
2. SIGHASH\_NONE:出力は署名されておらず、どんな値にも指定できます。このモードを使うと、「もし参加者全員が自らのお金を投資していれば、自分は投資してもかまわない。結果の金額がどうなるかは関係ない」となります。このモードを用いると、入力の連番を更新することで、各自が取引内容を更新できます。
3. SIGHASH\_SINGLE:SIGHASH\_NONEと同様に、入力は署名されていますが、連番は空のままですので、他の者が取引の新バージョンを作成できます。しかし、署名される出力は一つだけで、入力と同じ位置にある出力です。これを用いて、「もし自分の出力が自分の指定どおりであれば、自分は投資してもかまわない。他の人間については気にならない」となります。

上記の3モードに、SIGHASH\_ANYONECANPAY変更子を組み合わせることもできます。それを行うと、指定された入力のみが署名され、他の入力は何でもよくなります。

スクリプトにはCHECKMULTISIG 命令コードを含めることもできます。命令コードはn-of-mチェックを行います：複数の公開鍵を用意し、有効な署名の必要数も指定します。署名数は、公開鍵の数より少なくても構いません。2つの署名が無い限り金を使うことはできない、と出力で要求することもできます。この要領です：

2 <pubkey1> <pubkey2> 2 CHECKMULTISIGVERIFY

契約を安全に作成するには、2つの主要な方法があります。

1. 取引は、部分的に完成した形態か、不完全な形態で、P2Pネットワークの外で流通します。
2. 2つの取引が用いられます。一方（契約）は作成されて署名されていますが、まだ配信されていません。もう一方の取引（支払）が、契約が合意に達した後に配信され、それに従って契約も配信されます。

この手順を踏むことで、当事者は自分たちが何に同意しているのかを確実に理解できます。

これらを総合すると、ブロックチェーンの上で新たな金融ツールを構築することができます。

## 例１：預金の入力：

ウェブサイト上にアカウントを登録したい（フォーラムやWikiなど）、と仮定します。サイトの運営者に対して自分は信用できる人物だ、と証明したいのですが、実績のある信用情報を持っていません。解決方法の一つは、ウェブサイトにお金を払うことで信用を得る方法です。しかし、もしアカウントを閉鎖すると、支払ったお金は戻ってこないでしょう。あるいは、サイトの運営者をあまり信用しておらず、彼らがそのお金を横領するかもしれない、と考えているかもしれません。あるいは、ある日突然サイトが消えてしまうかもしれません。

自分はスパムではないと証明するために、サイト運営者に対して何からの支払を行いたいのですが、サイト運営者はそのお金を使い込むことはできない、というのが課題です。たとえサイト運営者が姿をくらませたとしても、支払ったお金は何もしなくても懐に戻るわけです。

この課題は、契約を利用して解決できます。

1. ユーザーとサイト運営者は、新規に作成した公開鍵をお互いに対して送ります。
2. ユーザーは取引Tx1（支払）を作成し、10 BTCを出力に指定します。その出力にはユーザーとサイト運営者両方の署名が必要です。取引は、まだ配信しません。サイトには、前回の鍵を利用します。
3. ユーザーはTx1のハッシュ結果をサイトに送ります。
4. サイトは取引Tx2（契約）を作成します。Tx2は金額Tx1を支払い、最初のステップでユーザーが提出したアドレス経由でユーザーに払い戻します。Tx1には2つの署名が必要なことに注意してください。ですので、この取引は完了しません。nLockTimeはかなり先の将来に設定してあります（例えば、6ヶ月先など）。入力の連番はゼロに設定してあります。
5. 最後に、未完了（署名が半分だけ行われた）取引は、ユーザーへ送り返されます。ユーザーは、契約内容が指定どおりであることを確認します。つまり、ビットコインがいずれは自分に返ってくるという意味です。ただし、想定外の事態が起こらないかぎり、6ヶ月待たなければなりません。連番がゼロのため、両者が合意すれば、契約は将来変更することもできます。入力に埋め込まれたスクリプトもまた、未完了です。ユーザーの署名があるべき場所には、ゼロと記載されています。ユーザーはゼロを署名で置き換えることで契約に署名し、入力を変更します。
6. ユーザーはTx1を配信し、続いてTx2を配信します。

この時点で、10BTCはユーザーとウェブサイトのどちらの側も、一方の独断では使用できない状態に置かれています。6ヶ月後、契約は満了し、ユーザーはお金を回収します。これは、ウェブサイトが消滅したとしても有効です。

ユーザーがアカウントをそれ以前に終了させたい、とした場合はどうでしょうか？ウェブサイトはTx2の新バージョンを作成します。その中でnLockTime をゼロにし、入力の連番をUINT\_MAXにしてから、再署名します。サイトはTxをユーザーに戻します。ユーザーはこれも署名します。ユーザーは次いで取引を配信し、これにより以前のバージョンの契約を破棄してビットコインを回収します。

6ヶ月の満了が間近なとき、ユーザーがアカウントを継続したいと希望した場合はどうでしょうか？同じ原理が働きます。再契約される契約には新しいnLockTimeが与えられ、連番は以前のバージョンから1つ加算され、2^32回再配信されます。何が起ころうとも、契約に携わる両者は契約変更に同意します。

当然のごとく、ユーザーに悪意があることが判明すれば（例：スパマー）、ウェブサイトは契約が満期以前に完了する事態には同意しません。ユーザーの悪意に基づく行為がひどい場合、預金の額を引き上げたり、契約期間が延長されたりします。

## 例2：エスクローと仲裁

買い手が誰かと取引を行いたいのですが、買い手は相手を知らないか、信用していません。通常は、取引がスムーズに済むので、第三者が介入する必要はありません。しかし、問題が発生すると、第三者に仲裁を依頼し、どちらの側が正しいかを判断してもらうでしょう。多分、プロの仲裁業者になるはずです。買い手と売り手のどちらの側から見ても、同じことを考えるだろうという点に注意してください。例えば、仲裁者は、売り手（業者）から出荷証明を要求するかもしれません。

言い換えると、当事者の一方はビットコインを閉じ込めておき、第三者の同意があって初めて使用できるようにしたいとします。

1. 買い手と売り手（業者）が、仲裁者を誰にするか同意します（例：ClearCoin）。
2. 業者に公開鍵を提出してもらいます（K1）。仲裁者に公開鍵を提出してもらいます（K2）。買い手（自分）の鍵を作成します（K３）。
3. 業者にK2を送ります。業者は仲裁者にワンタイムパスワードを生成して送りつけます。仲裁者はワンタイムパスワードをK2の秘密鍵を用いて解き、業者の持っている鍵が本物であることを証明します。
4. 買い手は以下の要領で出力スクリプトを記述し、取引（Tx1）を作成して配信します。

2 <K1> <K2> <K3> 3 CHECKMULTISIGVERIFY

この時点で、ビットコインはロック（閉じ込め）されています。このビットコインを使用するためには、以下の手順を踏まなければなりません。

1. 買い手と業者が合意します（取引が成立したか、業者が買い手に対して仲裁者抜きで払い戻します）。
2. 買い手と仲裁者が合意します（取引が失敗したか、支払拒否などの問題で仲裁者が買い手側に付いた場合）。
3. 仲裁者と業者が合意します（商品が届けられたか、買い手と業者が揉めている状態で業者が買い手のビットコインを回収した場合）。

入力に署名する際、その内容は接続先の出力となっています。従って、この取引を払い戻そうとすると、買い手はscriptSigを作成して、もう一方の署名が存在する場所にゼロを記入します。買い手はそれに署名し、スロットの一つを買い手自身の新しい署名に設定します。この取引は部分的に完了した形になっており、業者または仲裁者に送られて、そこで2つ目の署名が行われます。

## 例3：保証契約

[保証契約](http://en.wikipedia.org/wiki/Assurance_contract) は [公共財](http://www.auburn.edu/~johnspm/gloss/public_goods)の作成に資金を出す方法です。公共財とは、一旦作成されれば、誰もが無料で利用できる資産です。一般的な例は灯台です。建設の必要性は誰もが認めるのですが、船員の個人的な努力では建設費用は高すぎます。他の競争相手の船員たちにも利用されてしまうからです。

解決法の一つは、全員が公共財の作成に向けて投資を約束することです。約束された額が作成コストを上回らない限り、約束は確定とされません。つまり、十分な数の人から賛同が得られなかった場合、誰もお金を払わなくてすみます。

保証契約を用いた資金集めにおいて、ビットコインが既存の決済システムより優れている例として、小額の約束を自動的に取り付けなければいけない例があります。たとえば、インターネットラジオ局の設立や [ウェブサイトの翻訳](https://bitcointalk.org/index.php?topic=67255.msg788110#msg788110)などです。ブラウザー拡張を援助したいとします。ウェブページの現在の言語を検出し、ユーザーの言語への翻訳の資金要請を配信します。十分な数の拡張ユーザーが同時にそのページに集まれば（つまり、アクセス数の多いサイトにリンクされていれば）、十分な資金提供の約束が集まり、高品質の翻訳を手がける企業に対して支払がかかります。翻訳が完成すると、自動的にユーザーのブラウザに実装されます。

これは、ビットコインでは以下のようにして実現できます。

1. ある起業家が新規アドレスを作成し、最低1000BTCが集まれば、商品を実際に作る、と宣言します。誰でも参加できます。
2. 投資に参加したい人々はそれぞれ新しい取引を作成し、宣言されたアドレスに自身のビットコインを送金します。ただし、配信は行いません。この取引は通常の取引と似ていますが、3つの点で異なります。まず、変更は受け付けられません。出力が指定と異なる場合は、自分自身のアドレスに送金することで、異なるサイズの出力を作成しなければなりません。次に、入力スクリプトはSIGHASH\_ALL | SIGHASH\_ANYONECANPAYを用いて署名されます。最後に、出力金額は1000BTCに設定されます。出力の金額が入力の金額と異なるため、これは正当な取引とはみなされません。この点に注意してください。
3. 取引は起業家のサーバーのハードディスクにアップロードされます。サーバーは約束されたビットコインの量をカウントしてアップデートします。
4. サーバーに十分な量のビットコインがたまると、個別の取引を一つにまとめて、新たな取引に集約します。新取引には出力が一つしかなく、宣言されたアドレスに対して送金するだけです。投資のために集まった各取引の出力と同じです。新取引の入力（複数）には、投資のために集まった約束が集められます。
5. 最終形の取引は配信され、宣言されたアドレスに対して集まったビットコインを送金します。

この仕組みは、プロトコルの特性をいくつか、うまく利用しています。まず、SIGHASH フラグを用いています。SIGHASH\_ALLがデフォルトであり、入力スクリプト以外は取引内容の全てが署名されていることを意味します。SIGHASH\_ANYONECANPAYは追加の修飾子であり、署名はそれが埋め込まれた入力に対してのみ有効であることを示します。他の入力は署名されておらず、したがって他の入力は何であっても構いません。

このフラグを組み合わせることで、他に入力が追加されても有効であり続ける署名が出来上がります。ただし、取引のほかの属性が変更されると、取引は無効になります。

第二の特性は、出力される金額が入力される金額より大きい取引は無効である（当たり前です）、というルールを逆手に利用します。つまり、起業家に対して入力金額が出力より小さい取引を送っても何の問題もないことになります。彼はビットコインを現金化できないので、他の入力を集めて、合計が出力金額以上になるようにしなければなりません。

SIGHASH\_ANYONECANPAYを使わずとも、保証契約を作成することは可能です。実際、取引を作成せずとも投資約束が回収されるよう、2段階のプロセスを利用することができます。つまり、合計金額が指定に達すると、各出資者用に個別に入力が用意された取引が作成され、書名を集めて回る仕組みですしかし、SIGHASH\_ANYONECANPAYを用いて、最後に一つにまとめるやり方のほうが便利でしょう。

保証契約は、次のブロックに対して [**ネットワークセキュリティ**](https://en.bitcoin.it/wiki/Funding_network_security) 用の資金を出すために用いることもできます。こうすれば、ブロック空間が十分に足りているとしても、採掘作業が継続するよう、資金を出すことができます。

このコンセプトはTabarrokが自身の文書で詳細に記述しています。 [「優性保証契約を通じて公共財を私的に援助する方法」](http://mason.gmu.edu/~atabarro/PrivateProvision.pdf).優性保証契約においては、契約が不履行（期限内に十分な資金提供の申し出が集まらない場合）の際に、起業家はそれまでに投資を約束した人々に対して謝礼を出すことができます。この種の契約においては、参加するだけで利益が出るようにしており、従ってプロジェクトに参加することが常に正しい選択肢となります。[ビットコインにおける優性保証契約の設計方法](https://en.bitcoin.it/wiki/Dominant_Assurance_Contracts) が提案されています。

## 例4：外部条件を利用する

スクリプトは、その設計上、純粋に関数です。外部サーバーをポーリングしたり、変更可能なステートをインポートすることはできません（攻撃者が、ブロックチェーンを掌握する手がかりになってしまいます）。さらに、スクリプト言語を用いて行える作業は非常に限られています。幸いなことに、他の方法で現実世界での取引を行うことはできます。

ある老人が自分の孫に遺産を残したい、とします。孫の18歳の誕生日か、老人が死去するか、どちらか先に到来した時に有効になります。

この問題を解くには、老人はまず遺産を自分自身に送ります。そうすると、指定された金額の出力が一つだけになります。次に、彼は孫の18歳の誕生日に期限が設定された取引を作成します。この取引は、孫が所有している別の鍵に対してビットコインを支払います。老人は取引に署名して、孫に取引を送ります。ただし、配信は行いません。これにより、18歳の誕生日の条件が満たされます。誕生日が来れば、孫は取引を配信して、コインを回収します。彼がそれ以前に回収しようとしても、できません。期限があまりにも未来に設定されていると、取引をメモリープールから外すノードも出てくるでしょう。

老人の死去、の条件はより複雑です。ビットコインノードは任意条件に頼れないため、ここでは *オラクル（預言者）に頼る必要があります。*オラクルは鍵のペアを持つサーバーを指し、ユーザーが指定した条件が真になると、取引に署名できます。

ここに例を記します。老人は金額を出力する取引を作成し、出力はこう記述します。

<hash> OP\_DROP 2 <sons pubkey> <oracle pubkey> CHECKMULTISIG

これが、オラクルに対するスクリプトとなります。データをスタックに押しやり、その後すぐに削除するので、通常のスクリプト形態とは異なります。Pubkey（公開鍵）はオラクルのウェブサイトで公開されており、誰もが見れます。Hashは、老人が死んだというユーザー条件を満たすハッシュ演算結果となり、オラクルが検証可能なように書かれています。例えば、以下の行のハッシュともなりえます。

if (has\_died('john smith', born\_on=1950/01/02)) return (10.0, 1JxgRXEHBi86zYzHN2U4KMyRCg4LvwNUrp);

この記述内容は例えです。オラクルが指定さえすれば、内容はなんとでもなります。戻り値は出力となります。金額と、孫が持っているアドレスで構成されています。

もう一度強調しますが、老人はこの取引を作成した後、配信せずに直接孫に渡します。老人はまた、取引内でハッシュ演算される記述も書き入れ、ロック解除できるオラクルも指定します。

次のアルゴリズムが用いられます。

1. オラクルは観察要請を受け取ります。要請には、ユーザーが提供する条件、出力スクリプトのコピー、ユーザーが提供する未完了の取引が含まれます。この取引は、scriptSigを除いて全てが完了しています。scriptSigには孫の署名だけが含まれており、それだけでは出力のロックを解除できません。
2. オラクルはユーザーが提供した条件ハッシュをチェックし、出力スクリプトの値と比較します。もしハッシュが条件に合わない場合、エラーが返されます。
3. オラクルは条件自体を検証します。結果、出力先アドレスに合致しない場合は、エラーが返されます。
4. エラーが出なければ、オラクルは取引に署名し、署名をユーザーに返します。ビットコイン取引に署名する際、入力スクリプトは接続されている出力スクリプトに設定されている点に注意してください。なぜかと言うと、OP\_CHECKSIGを実行する際、命令コードを含んでいるスクリプトは、署名自体を含むスクリプト内ではなく、評価対象の入力に組み込まれているからです。オラクルは、署名を要請されている出力を全て見ることはなく、見る必要もありません。オラクルは出力スクリプトを見ており、自身の公開鍵を持っており、ユーザーが提供した条件のハッシュ関数を得ており、それだけあれば出力スクリプトをチェックして取引を完了させるには十分です。
5. ユーザーは新しい署名を受け取り、それをscriptSig に入力し、取引を配信します。

老人が死去したとオラクルが同意した場合のみ、孫は2つの取引を配信でき（契約と、請求です）、コインを回収できます。

オラクルは理論的にあらゆるものを検証できます。それでも、ブロックチェーン内の出力スクリプト形式は同じままです。次の可能性を考慮してみてください。

today() == 2011/09/25 && exchange\_rate(mtgoxUSD) >= 12.5 && exchange\_rate(mtgoxUSD) <= 13.5 指定日における交換レートが2つの値の間に収まることを条件とする google\_results\_count(site:www.google.com/hostednews 'Mike Hearn' olympic gold medal) > 0 A bet on me doing something that I will never actually do // ユーロビジョンコンテストの優勝者を二人の候補の中から当てる. if (eurovision\_winner() == 'Azerbaijan') return 1Lj9udBVDwptFffGSJSC2sohCfudQgSTPD; else return 1JxgRXEHBi86zYzHN2U4KMyRCg4LvwNUrp;

オラクルが署名を行う条件を複雑にしても、ブロックチェーン内に入力されるのはハッシュ一つだけで済みます。

### 信用を不要にする：課題

オラクル自身の信用度を減らす方法はいくつかあります。

最初の例に戻ると、オラクルは孫が解除しようとしている取引を見ていません（配信されていないからです）。ゆえに、オラクル自身は孫に対して取引を人質に脅迫することができません（署名対象の取引が存在しているかどうかすら、わからないのですから）。この仕組みを利用する人は、定期的にオラクルに **課題を突きつけ** 、オラクルが指定どおりの出力を出せるかどうか確認する必要があります。課題自体は、署名対象のTxが無効（接続先の取引は存在しません）なので、ビットコインを使用せずに出せます。オラクルにとっては、署名を要求されている内容が本物か、でたらめかを知る手立てはありません。まだ満たされていない条件を使ってオラクルを試すのは、まだ研究段階です。

### 信用を不要にする：複数の独立したオラクル

CHECKMULTISIG内のキー数を増やして、必要であれば **n-of-m** オラクルを設定できます。もちろん、オラクル同士が真に独立しており、衝突の可能性がないことをチェックしなければなりません。

### 信用を不要にする：信頼性の高いハードウェア

一般的に入手可能なハードウェアを用いて、Intel TXTやAMD の同等品 (SKINIT)の **高信頼性処理** を行い、ブラックボックスの閉じられたハードウェア環境を作成し、TPMチップを利用して第三者に認証してもらうことができます。第三者は、ハードウェアが指定された状態にあることを検証します。この状態を打ち崩すためには、CPU上でのみ完結して動作するプログラムに改ざんしなければなりません。極端なケースでは、メモリーバスにデータが接触すらしない必要があります（プログラムが十分に小さければ、オンダイキャッシュの上だけで走らせることは可能です）。

### 信用を不要にする：アマゾンAWSオラクル

最後に、現時点で最も実用的な方法は、アマゾンウェブサービスを利用することです。2013年11月時点において、フル稼働するオラクルに最も近いのは [アマゾンウェブサービスを用いて高信頼性処理環境を作成する方法](https://bitcointalk.org/index.php?topic=301538.0)であり、 [選択性SSLログインおよび解読プロジェクトの過程で生み出されました](https://bitcointalk.org/index.php?topic=173220.0)。この仕組みは、アマゾンを信頼の基本に置き、アマゾンのAPIを用いてオラクルを信頼できる形で検証し、暗号化されたSSLセッションをオンラインバンクインターフェースに記録します。個人間のやりとりに問題が生じた場合、ログを解読すれば問題を解決できます。

## 例5：チェーンを超えて取引を行う場合

ビットコイン技術を応用して、 [相互に独立した通貨を複数生み出すことができます](https://en.bitcoin.it/wiki/Alternative_chain)。Namecoinはそうした通貨の一例であり、運営ルールはビットコインとは少し異なります。Namecoinを利用して、namespace（名前空間）で名前を借りることができます。ビットコインと同じ理念を体現する通貨は、多少の信用があれば、お互いに取引を行えます。

例えば、ある企業共同体が、共同体の銀行口座内の預金に1:1で担保された暗号化通貨であるEURコインを発行しているとします。そうした通貨はビットコインとは異なる利点を持ちます。ビットコインより中央集権的ですが、為替リスクが存在しません。ビットコインとEURコイン同士で交換したい人々がいるとします。企業共同体がかかわってくるのは、通常の銀行口座の預金を出し入れしなければいけない場合だけです。

これを実装する場合は、 [TierNolanが提唱したプロトコルが](https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949) 用いられます：

1. 当事者「A」は、ランダムなデータ「ｘ」を生成します（データ内容は秘密です）。
2. 当事者「A」は、支払「Tx1」を生成します。この支払の出力の一つには、チェーン間取引のスクリプトが埋め込まれています。以下、このスクリプトとその解説を見ていきましょう。2つの鍵(鍵「A」と鍵「B」）あるいは（秘密「ｘ」と鍵「B」）をもって署名すれば、コインが回収できます。この取引は配信されていません。チェーンはハッシュを含むスクリプトをリリースします（秘密自体はリリースしません）。
3. 当事者「A」はTx2（契約）を生成します。ここでは支払「Tx1」が行われ、出力の一つは鍵「A」に戻ります。将来の期限付きであり、入力にはゼロではじめる連番があるため、入れ替え可能です。「A」はTx２に署名し、「B」に送ります。「B」もTx2に署名し、送り返します。
4. 「A」はTx1とTx2を配信します。当事者「B」はコインを見ることができますが、「B」へ向かう出力がないために、使うことができません。それに、Txはまだ確定していません。
5. 「B」はおなじ手順を逆にして、代替チェーンの上で実行します。「B」の期限は、「A」の期限よりずっと長くとる必要があります。取引の両者共に、この時点では確定待ちです。まだ完了していません。
6. 「A」は秘密を知っているために、「A」は直ちにコインを回収できます。ただし、「A」はコイン回収の過程で、秘密「ｘ」を「B」に公開することになります。「B」はその秘密「ｘ」を用いて、取引のもう一方（「ｘ」と鍵「B」）を完了させます。

このプロトコルを用いると、完全にピアツーピアに沿った形で取引を自動的に行えます。これは、流動性が高くなります。

チェーン間取引のスクリプトは以下のようになります。

IF 2 <key A> <key B> 2 CHECKMULTISIGVERIFY ELSE <key B> CHECKSIGVERIFY SHA256 <hash of secret x> EQUALVERIFY ENDIF

契約の入力スクリプトは以下のどれかになります。

<sig A> <sig B> 1

または

<secret x> <sig B> 0

つまり、最初のデータ要素によって、どちらを利用するかが決まります。

詳細は、 [アトミックなチェーン間取引](https://en.bitcoin.it/wiki/Atomic_cross-chain_trading) を参照してください。

EURコインは自然な発想ですが、ピアツーピア通貨取引（ビットコインと法定通貨の取引）を実装する方法は他にもあります。詳細は [Ripple通貨取引](https://en.bitcoin.it/wiki/Ripple_currency_exchange) を参照してください。

Sergio Demian-Lernerは [P2PTradeX](https://bitcointalk.org/index.php?topic=91843.0)を提案しました。一方のブロックチェーンの検証ルールを、他方のブロックチェーン内に埋め込むことを盛り込んだソリューションです。

## 例6：検証後の支払：純粋な関数であれば、解決可能です

例4において、任意のプログラムの出力結果に応じて支払を行う方法を探りました。こうしたプログラムは非常に強力で、一般的なプログラムが行えること（例えば、ウェブページの取り込み）などは全て行えます。欠点は、第三者（オラクル）が必要な点です。オラクルの持つ信用度に頼る度合いを減らすテクニックはありますが、全てなくすことはできません。

純粋に関数で構成されているような一部のプログラムについては、新たな暗号化テクニックが利用可能になってきています。第三者が必要ないように、信用に全く頼る必要がなくなるものです。こうしたプログラムはI/O操作は行えませんが、そうした制限は多くの場合関係ないか、他の方法で代替できます。例えば、プログラム自身でドキュメントをダウンロードするのではなく、署名済みでタイムスタンプが押されたドキュメントを入力として用いる、などです。

このプロトコルの説明はこちらで読めます。[User:Gmaxwell/why\_hash\_locked](https://en.bitcoin.it/wiki/User:Gmaxwell/why_hash_locked)

## 例7：あらかじめ指定された相手に対して（マイクロ）決済の額を細かく調整

ビットコイン取引はこれまでの決済システムと比較すると非常に安く行えますが、採掘と保管を行うために、どうしても多少のコストはかかります。取引配信のコストをかけずに、特定の相手に対して送金する額を頻繁に細かく調整したい場合があります。

たとえば、初めて利用するカフェのワイヤレスアクセスなど、まだ信用が確立されていないインターネットアクセスポイントを例にとってみましょう。10キロバイトの利用ごとに0.001BTCを払いたい、とします。ただし、カフェのシステム内でアカウントを作成せずに、です。全く信頼に基づかないソリューションの場合、全自動で処理が行われます。つまり、スマートフォンのモバイル財布内に、月初めにある程度のビットコインを入れておき、その後はスマートフォンが必要に応じて自動的にインターネットアクセスの費用を決済します。カフェ側にしても、だまされる心配なく、誰にでも利用してもらうほうがありがたいでしょう。

このためには、次のプロトコルが利用されます。このプロトコルにおいては、nLockTimeの挙動が元々の設計とは **異なる** ので注意してください。2013年あたりを契機に、期限付きの取引は標準でなくなり、メモリープールに入力されなくなりました。つまり、起源が切れる前に配信することができなくなったのです。nLockTimeが元々のサトシの設計仕様に戻されたとき、このプロトコルの派生物が必要になりました。それが、以下に紹介する内容です。

クライアントは送金を行い、サーバーはそれを受け取るとします。以下、クライアント側の視点から記述します。

1. 公開鍵を作成します（K1）。公開鍵をサーバーから要求します（K2）。
2. 取引（T1）を作成して署名しますが、まだ配信はしません。この取引は（例えば）10BTCの支払を行い、サーバーの公開鍵と自分（クライアント）の公開鍵の両方を出力に用います。OP\_CHECKMULTISIGを用いるとうまくいきます。使用される金額は、効率性とのトレードオフとなります。
3. T1の出力に払い戻し取引（T2）を作成して接続し、T1の出力金額が全て自分に戻ってくるようにします。将来の期限付きです。例えば、数時間とします。まだ署名してはいけません。未署名の取引をサーバーへ送ります。通例に従うと、出力スクリプトは「2 K1 K2 2 CHECKMULTISIG」となります。
4. サーバーは公開鍵K2を利用してT2を署名し、署名をクライアントに返します。この時点では、T1は見られていません。ハッシュ（未署名のT2）のみが見られている点に注意してください。
5. クライアントはサーバーの署名が正式であることを確認します。そうでないばあいは、取引を中止します。
6. クライアントはT1を署名し、署名をサーバーに送ります。この時点で、サーバーは取引を配信します（サーバーとクライアントが共にネットワーク接続されていれば、どちらの側でも配信を行えます）。これで、金額は確定します。
7. クライアントは次に、新規取引のT3を作成します。これは払い戻し取引同様にT1に接続され、2つの出力を持ちます。出力の片方はK1に行き、もう片方はK2に行きます。まず、全ての金額が最初の出力（K1）に集められます。つまり、払い戻し取引と同じなのですが、今度は期限がありません。クライアントはT3を署名し、取引と署名をサーバーに送ります。
8. サーバーでは、自身に送られた出力が指定どおりの内容であることと、クライアントが行った署名が正しいことを確認します。
9. クライアントがサーバーに送金したい場合は、T3を複製してサーバーの出力金額を増やし、自身の金額を減らすよう調整します。次に、クライアントは調整（増額）済みT3を再度署名し、サーバーに署名を送ります。取引全てを送ることはしません。署名と、増額内容だけで十分です。サーバーは自身が持つT3の金額を調整して、署名を検証します。

この金額調整のやりとりはセッションが終了するか、1日の期限が終わりに近づくまで続きます。その時点で、APは最後の調整取引に署名して配信し、最終金額を回収します。サーバーが消失したり停止したりして金額が宙に浮いてしまった場合は、払い戻し取引を用いて事態を収束させます。この事態が発生すると、期限に達した時点でクライアントは払い戻し取引を配信し、金額を全て回収します。

このプロトコルは [ビットコインに実装されました](https://code.google.com/p/bitcoinj/wiki/WorkingWithMicropayments)。

nLockTimeを用いた取引がメモリープールに（再度）入力され、代替取引が有効になる際には、プロトコルの派生物を用いる必要があります。この場合、払い戻し取引は利用されません。代わりに、各T3には前回よりも一つ番号が大きい連番が付けられ、全てのT3には上記と同じ期限が設定されます。支払が行われる度に連番が加算され、これにより最新バージョンが常に優先されます。チャンネルプロトコルがきれいに閉じられていない場合、期限に達するまでは送金は確定しません。この事態を防ぐため、取引の両方の当事者がT3の連番を0xFFFFFFFFに設定して署名すれば、nLockTimeの値に関係なく、直ちに確定が得られます。

T期限と連番を利用すると、以下の攻撃を防げます。APは接続機能を提供し、ユーザーはTX2の最初のバージョンを用いて出力を自分自身で [二重使用](https://en.bitcoin.it/wiki/Double-spending) します。これで、カフェは金額を徴収できなくなります。ユーザーが二重使用を試みると、TXが正しく組み込まれなくなり、アクセスポイントはその間TXの配信を観察する時間の余裕が生まれます。その結果、最後のバージョンを配信することになり、二重使用の試みを上書きして無効にできます。

後半に紹介したプロトコルは取引の入れ替えをベースにしており、前半より柔軟です。理由は、クライアントがサーバーからいつでも署名を受け取れるのであれば、チャンネルの有効期間内に金額を上げるだけでなく下げることもできるからです。ただ、多くの場合ではこうした機能は特に必要とされていません。入れ替えにより、2者間で行うよりも複雑なチャンネル構成が可能になります。そうしたケースの利用法は、読者の課題としておきましょう。

## 例8：複数のグループが参加する分散型賭博

例6で使用されたテクニックを応用して、高度なスクリプト言語を追加して、胴元が存在せず、みんなが参加できる分散型賭博を作成できます。ここで利用するプロトコルは次の文書で紹介されています。 [「ビットコイン上で複数の参加者が演算に参加する方法」](http://eprint.iacr.org/2013/784).もっと短い解説が、将来このWikiに掲載されるかもしれません。

## 参考

* [スクリプト](https://en.bitcoin.it/wiki/Script)

[カテゴリ](https://en.bitcoin.it/wiki/Special:Categories):

* [技術用語](https://en.bitcoin.it/wiki/Category:Technical)